Aries Cloud Agent Python Documentation

Nicholas Rempel, Andrew Whitehead

Aug 15, 2019

Aries Cloud Agent Python - Modules

CHAPTER 1

aries_cloudagent package

Aries Cloud Agent.

1.1 Subpackages

1.1.1 aries_cloudagent.admin package

Subpackages

aries_cloudagent.admin.tests package

Submodules

aries_cloudagent.admin.tests.test_admin_server module

class aries_cloudagent.admin.tests.test_admin_server.**TestAdminServerApp**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

get_admin_server() → aries_cloudagent.admin.server.AdminServer

get_application()
Override the get_app method to return your application.

outbound_message_router(**args*)

test_index()

test_start_bad_settings()

test_start_stop()

test_status()

test_swagger()

test_websocket()

class aries_cloudagent.admin.tests.test_admin_server.**TestAdminServerWebhook**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

get_admin_server() → aries_cloudagent.admin.server.AdminServer

get_application()
Override the get_app method to return your application.

outbound_message_router(**args*)

receive_hook(*request*)

setUpAsync()

test_webhook()

Submodules

aries_cloudagent.admin.base_server module

Abstract admin server interface.

class aries_cloudagent.admin.base_server.**BaseAdminServer**
Bases: abc.ABC

Admin HTTP server class.

add_webhook_target(*target_url: str*, *topic_filter: Sequence[str] = None*, *retries: int = None*)
Add a webhook target.

remove_webhook_target(*target_url: str*)
Remove a webhook target.

send_webhook(*topic: str*, *payload: dict*)
Add a webhook to the queue, to send to all registered targets.

start() → None
Start the webserver.

Raises AdminSetupError – If there was an error starting the webserver

stop() → None
Stop the webserver.

aries_cloudagent.admin.error module

Admin error classes.

exception aries_cloudagent.admin.error.**AdminError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Base class for Admin-related errors.

exception aries_cloudagent.admin.error.**AdminSetupError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.admin.error.AdminError*

Admin server setup or configuration error.

aries_cloudagent.admin.routes module

Register default routes.

aries_cloudagent.admin.routes.**register_module_routes**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a03db00>*)

Register default routes with the webserver.

Eventually this should use dynamic registration based on the currently-selected message families.

aries_cloudagent.admin.server module

Admin server classes.

class aries_cloudagent.admin.server.**AdminModulesSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Schema for the modules endpoint.

result

Used by autodoc_mock_imports.

class aries_cloudagent.admin.server.**AdminResponder**(*send: Coroutine[T_co, T_contra, V_co], webhook: Coroutine[T_co, T_contra, V_co], **kwargs*)

Bases: *aries_cloudagent.messaging.responder.BaseResponder*

Handle outgoing messages from message handlers.

send_outbound(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)

Send outbound message.

Parameters **message** – The *OutboundMessage* to be sent

send_webhook(*topic: str*, *payload: dict*)

Dispatch a webhook.

Parameters

- **topic** – the webhook topic identifier
- **payload** – the webhook payload value

class aries_cloudagent.admin.server.**AdminServer**(*host: str, port: int, context: aries_cloudagent.config.injection_context.InjectionContext, outbound_message_router: Coroutine[T_co, T_contra, V_co]*)

Bases: *aries_cloudagent.admin.base_server.BaseAdminServer*

Admin HTTP server class.

add_webhook_target(*target_url: str*, *topic_filter: Sequence[str] = None*, *retries: int = None*)

Add a webhook target.

complete_webhooks()

Wait for all pending webhooks to be dispatched, used in testing.

make_application() → <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a0e4128>

Get the aiohttp application instance.

modules_handler(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a0e4128>*)
Request handler for the loaded modules list.

Parameters **request** – aiohttp request object

Returns The module list response

on_startup(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a0e4128>*)
Perform webserver startup actions.

redirect_handler(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a0e4128>*)
Perform redirect to documentation.

remove_webhook_target(*target_url: str*)
Remove a webhook target.

send_webhook(*topic: str*, *payload: dict*)
Add a webhook to the queue, to send to all registered targets.

start() → None
Start the webserver.

Raises AdminSetupError – If there was an error starting the webserver

status_handler(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a0e4128>*)
Request handler for the server status information.

Parameters **request** – aiohttp request object

Returns The web response

status_reset_handler(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a0e4128>*)
Request handler for resetting the timing statistics.

Parameters **request** – aiohttp request object

Returns The web response

stop() → None
Stop the webserver.

websocket_handler(*request*)
Send notifications to admin client over websocket.

class aries_cloudagent.admin.server.**AdminStatusSchema**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Schema for the status endpoint.

class aries_cloudagent.admin.server.**WebhookTarget**(*endpoint: str*, *topic_filter: Sequence[str] = None*, *retries: int = None*)
Bases: object

Class for managing webhook target information.

topic_filter
Accessor for the target's topic filter.

1.1.2 aries_cloudagent.cache package

Subpackages

aries_cloudagent.cache.tests package

Submodules

aries_cloudagent.cache.tests.test_basic_cache module

class aries_cloudagent.cache.tests.test_basic_cache.**TestBasicCache**
Bases: object

test_flush(*cache*)

test_get_none(*cache*)

test_get_valid(*cache*)

test_set_dict(*cache*)

test_set_expires(*cache*)

test_set_str(*cache*)

aries_cloudagent.cache.tests.test_basic_cache.**cache**()

Submodules

aries_cloudagent.cache.base module

Abstract base classes for cache.

class aries_cloudagent.cache.base.**BaseCache**
Bases: abc.ABC

Abstract cache interface.

clear(*key: str*)
Remove an item from the cache, if present.

Parameters **key** – the key to remove

flush()
Remove all items from the cache.

get(*key: str*)
Get an item from the cache.

Parameters **key** – the key to retrieve an item for

Returns The record found or *None*

set(*key: str*, *value: Any*, *ttl: int*)
Add an item to the cache with an optional ttl.

Parameters

- **key** – the key to set an item for
- **value** – the value to store in the cache
- **ttl** – number of second that the record should persist

aries_cloudagent.cache.basic module

Basic in-memory cache implementation.

class aries_cloudagent.cache.basic.**BasicCache**
Bases: *aries_cloudagent.cache.base.BaseCache*

Basic in-memory cache class.

clear(*key: str*)
Remove an item from the cache, if present.

Parameters **key** – the key to remove

flush()
Remove all items from the cache.

get(*key: str*)
Get an item from the cache.

Parameters **key** – the key to retrieve an item for

Returns The record found or *None*

set(*key: str*, *value: Any*, *ttl: int = None*)
Add an item to the cache with an optional ttl.

Overwrites existing cache entries.

Parameters

- **key** – the key to set an item for
- **value** – the value to store in the cache
- **ttl** – number of seconds that the record should persist

1.1.3 aries_cloudagent.config package

Subpackages

aries_cloudagent.config.tests package

Submodules

aries_cloudagent.config.tests.test_injection_context module

class aries_cloudagent.config.tests.test_injection_context.**TestInjectionContext**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

setUp()

test_inject_scope()
Test a scoped injection.

test_inject_simple()
Test a basic injection.

test_settings_init()
Test settings initialization.

test_settings_scope()
Test scoped settings.

test_simple_scope()
Test scope entrance and exit.

aries_cloudagent.config.tests.test_injector module

class aries_cloudagent.config.tests.test_injector.**MockInstance**(*value*, ***kwargs*)
Bases: object

open()

class aries_cloudagent.config.tests.test_injector.**MockProvider**(*value*)
Bases: *aries_cloudagent.config.base.BaseProvider*

provide(*settings: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Provide the object instance given a config and injector.

class aries_cloudagent.config.tests.test_injector.**TestInjector**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

setUp()

test_bad_provider()
Test empty and invalid provider results.

test_inject_cached()
Test a provider class injection.

test_inject_class()
Test a provider class injection.

test_inject_class_dependency()
Test a provider class injection with a dependency.

test_inject_class_name()
Test a provider class injection with a named class.

test_inject_provider()
Test a provider injection.

test_inject_simple()
Test a basic injection.

test_settings_init()
Test settings initialization.

aries_cloudagent.config.tests.test_settings module

class aries_cloudagent.config.tests.test_settings.**TestSettings**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_get_formats()
Test retrieval with formatting.

test_remove()
Test value removal.

test_set_default()
Test default value.

test_settings_init()
Test settings initialization.

Submodules

aries_cloudagent.config.base module

Configuration base classes.

class aries_cloudagent.config.base.**BaseInjector**
Bases: abc.ABC

Base injector class.

copy() → aries_cloudagent.config.base.BaseInjector
Produce a copy of the injector instance.

inject(*base_cls: type*, *settings: Mapping[str, object] = None*, *, *required: bool = True*) → object
Get the provided instance of a given class identifier.

Parameters

- **cls** – The base class to retrieve an instance of
- **settings** – An optional mapping providing configuration to the provider

Returns An instance of the base class, or None

class aries_cloudagent.config.base.**BaseProvider**
Bases: abc.ABC

Base provider class.

provide(*settings: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Provide the object instance given a config and injector.

class aries_cloudagent.config.base.**BaseSettings**
Bases: collections.abc.Mapping, typing.Generic

Base settings class.

copy() → aries_cloudagent.config.base.BaseSettings
Produce a copy of the settings instance.

extend(*other: Mapping[str, object]*) → aries_cloudagent.config.base.BaseSettings
Merge another mapping to produce a new settings instance.

get_bool(**var_names*, *default=None*) → bool
Fetch a setting as a boolean value.

Parameters

- **var_names** – A list of variable name alternatives
- **default** – The default value to return if none are defined

get_int(**var_names*, *default=None*) → int
Fetch a setting as an integer value.

Parameters

- **var_names** – A list of variable name alternatives
- **default** – The default value to return if none are defined

get_str(**var_names*, *default=None*) → str
Fetch a setting as a string value.

Parameters

- **var_names** – A list of variable name alternatives
- **default** – The default value to return if none are defined

get_value(**var_names*, *default=None*)
Fetch a setting.

Parameters

- **var_names** – A list of variable name alternatives
- **default** – The default value to return if none are defined

Returns The setting value, if defined, otherwise the default value

exception aries_cloudagent.config.base.**ConfigError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.error.BaseError*

A base exception for all configuration errors.

exception aries_cloudagent.config.base.**InjectorError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.config.base.ConfigError*

The base exception raised by *BaseInjector* implementations.

exception aries_cloudagent.config.base.**ProviderError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.config.base.ConfigError*

The base exception raised by *BaseProvider* implementations.

exception aries_cloudagent.config.base.**SettingsError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.config.base.ConfigError*

The base exception raised by *BaseSettings* implementations.

aries_cloudagent.config.injection_context module

Injection context implementation.

class aries_cloudagent.config.injection_context.**InjectionContext**(**, settings: Mapping[str, object] = None, enforce_typing: bool = True*)
Bases: *aries_cloudagent.config.base.BaseInjector*

Manager for configuration settings and class providers.

ROOT_SCOPE = 'application'

copy() → aries_cloudagent.config.injection_context.InjectionContext
Produce a copy of the injector instance.

inject(*base_cls: type*, *settings: Mapping[str, object] = None*, *, *required: bool = True*) → object
Get the provided instance of a given class identifier.

Parameters

- **cls** – The base class to retrieve an instance of
- **settings** – An optional mapping providing configuration to the provider

Returns An instance of the base class, or None

injector
Accessor for scope-specific injector.

injector_for_scope(*scope_name: str*) → aries_cloudagent.config.injector.Injector
Fetch the injector for a specific scope.

Parameters **scope_name** – The unique scope identifier

scope_name
Accessor for the current scope name.

settings
Accessor for scope-specific settings.

start_scope(*scope_name: str*, *settings: Mapping[str, object] = None*) → aries_cloudagent.config.injection_context.InjectionContext
Begin a new named scope.

Parameters

- **scope_name** – The unique name for the scope being entered
- **settings** – An optional mapping of additional settings to apply

Returns A new injection context representing the scope

update_settings(*settings: Mapping[str, object]*)
Update the scope with additional settings.

exception aries_cloudagent.config.injection_context.**InjectionContextError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.config.base.InjectorError*

Base class for issues in the injection context.

class aries_cloudagent.config.injection_context.**Scope**(*name*, *injector*)
Bases: tuple

injector
Alias for field number 1

name
Alias for field number 0

aries_cloudagent.config.injector module

Standard Injector implementation.

class aries_cloudagent.config.injector.**Injector**(*settings: Mapping[str, object] = None*, *enforce_typing: bool = True*)

Bases: *aries_cloudagent.config.base.BaseInjector*

Injector implementation with static and dynamic bindings.

bind_instance(*base_cls: type*, *instance: object*)
Add a static instance as a class binding.

bind_provider(*base_cls: type*, *provider: aries_cloudagent.config.base.BaseProvider*, ***, *cache: bool = False*)
Add a dynamic instance resolver as a class binding.

clear_binding(*base_cls: type*)
Remove a previously-added binding.

copy() → aries_cloudagent.config.base.BaseInjector
Produce a copy of the injector instance.

get_provider(*base_cls: type*)
Find the provider associated with a class binding.

inject(*base_cls: type*, *settings: Mapping[str, object] = None*, ***, *required: bool = True*)
Get the provided instance of a given class identifier.

Parameters

- **cls** – The base class to retrieve an instance of
- **params** – An optional dict providing configuration to the provider

Returns An instance of the base class, or None

settings
Accessor for scope-specific settings.

aries_cloudagent.config.provider module

Service provider implementations.

class aries_cloudagent.config.provider.**CachedProvider**(*provider: aries_cloudagent.config.base.BaseProvider*)

Bases: *aries_cloudagent.config.base.BaseProvider*

Cache the result of another provider.

provide(*config: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Provide the object instance given a config and injector.

class aries_cloudagent.config.provider.**ClassProvider**(*instance_cls: Union[str, type]*, **ctor_args*, *async_init: str = None*, ***ctor_kwargs*)

Bases: *aries_cloudagent.config.base.BaseProvider*

Provider for a particular class.

class Inject(*base_cls: type*)
Bases: object

A class for passing injected arguments to the constructor.

provide(*config: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Provide the object instance given a config and injector.

class aries_cloudagent.config.provider.**InstanceProvider**(*instance*)
Bases: *aries_cloudagent.config.base.BaseProvider*

Provider for a previously-created instance.

provide(*config: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Provide the object instance given a config and injector.

class aries_cloudagent.config.provider.**StatsProvider**(*provider: aries_cloudagent.config.base.BaseProvider, methods: Sequence[str], *, ignore_missing: bool = True*)
Bases: *aries_cloudagent.config.base.BaseProvider*

Add statistics to the results of another provider.

provide(*config: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Provide the object instance given a config and injector.

aries_cloudagent.config.settings module

Settings implementation.

class aries_cloudagent.config.settings.**Settings**(*values: Mapping[str, object] = None*)
Bases: *aries_cloudagent.config.base.BaseSettings*

Mutable settings implementation.

clear_value(*var_name: str*)
Remove a setting.

Parameters **var_name** – The name of the setting

copy() → aries_cloudagent.config.base.BaseSettings
Produce a copy of the settings instance.

extend(*other: Mapping[str, object]*) → aries_cloudagent.config.base.BaseSettings
Merge another settings instance to produce a new instance.

get_value(**var_names*, *default=None*)
Fetch a setting.

Parameters

- **var_names** – A list of variable name alternatives
- **default** – The default value to return if none are defined

set_default(*var_name: str*, *value*)
Add a setting if not currently defined.

Parameters

- **var_name** – The name of the setting
- **value** – The value to assign

set_value(*var_name: str*, *value*)
Add a setting.

Parameters

- **var_name** – The name of the setting
- **value** – The value to assign

1.1.4 aries_cloudagent.holder package

Subpackages

aries_cloudagent.holder.tests package

Submodules

aries_cloudagent.holder.tests.test_indy module

Submodules

aries_cloudagent.holder.base module

Base holder class.

class aries_cloudagent.holder.base.**BaseHolder**
Bases: abc.ABC

Base class for holder.

aries_cloudagent.holder.indy module

Indy issuer implementation.

class aries_cloudagent.holder.indy.**IndyHolder**(*wallet*)
Bases: *aries_cloudagent.holder.base.BaseHolder*

Indy holder class.

create_credential_request(*credential_offer*, *credential_definition*, *did*)
Create a credential offer for the given credential definition id.

Parameters

- **credential_offer** – The credential offer to create request for
- **credential_definition** – The credential definition to create an offer for

Returns A credential request

create_presentation(*presentation_request: dict*, *requested_credentials: dict*, *schemas: dict*, *credential_definitions: dict*)
Get credentials stored in the wallet.

Parameters

- **presentation_request** – Valid indy format presentation request
- **requested_credentials** – Indy format requested_credentials

- **schemas** – Indy formatted schemas_json
- **credential_definitions** – Indy formatted schemas_json

delete_credential (*credential_id: str*)
Remove a credential stored in the wallet.

Parameters **credential_id** – Credential id to remove

get_credential (*credential_id: str*)
Get a credential stored in the wallet.

Parameters **credential_id** – Credential id to retrieve

get_credentials (*start: int*, *count: int*, *wql: dict*)
Get credentials stored in the wallet.

Parameters

- **start** – Starting index
- **count** – Number of records to return
- **wql** – wql query dict

get_credentials_for_presentation_request_by_referent (*presentation_request: dict, referents: Sequence[str], start: int, count: int, extra_query: dict = {}*)
Get credentials stored in the wallet.

Parameters

- **presentation_request** – Valid presentation request from issuer
- **referents** – Presentation request referents to use to search for creds
- **start** – Starting index
- **count** – Maximum number of records to return
- **extra_query** – wql query dict

store_credential (*credential_definition*, *credential_data*, *credential_request_metadata*)
Store a credential in the wallet.

Parameters

- **credential_definition** – Credential definition for this credential
- **credential_data** – Credential data generated by the issuer

1.1.5 aries_cloudagent.issuer package

Submodules

aries_cloudagent.issuer.base module

Ledger issuer class.

class aries_cloudagent.issuer.base.**BaseIssuer**
Bases: abc.ABC

Base class for issuer.

aries_cloudagent.issuer.indy module

Indy issuer implementation.

class aries_cloudagent.issuer.indy.**IndyIssuer**(*wallet*)
Bases: *aries_cloudagent.issuer.base.BaseIssuer*

Indy issuer class.

create_credential(*schema*, *credential_offer*, *credential_request*, *credential_values*)
Create a credential.

Args schema: Schema to create credential for credential_offer: Credential Offer to create credential for credential_request: Credential request to create credential for credential_values: Values to go in credential

Returns A tuple of created credential, revocation id

create_credential_offer(*credential_definition_id*)
Create a credential offer for the given credential definition id.

Parameters **credential_definition_id** – The credential definition to create an offer for

Returns A credential offer

exception aries_cloudagent.issuer.indy.**IssuerError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.error.BaseError*

Generic issuer error.

aries_cloudagent.issuer.util module

Issuer utils.

aries_cloudagent.issuer.util.**encode**(*orig: Any*) → str
Encode a credential value as an int.

Encode credential attribute value, purely stringifying any int32 and leaving numeric int32 strings alone, but mapping any other input to a stringified 256-bit (but not 32-bit) integer. Predicates in indy-sdk operate on int32 values properly only when their encoded values match their raw values.

Parameters **orig** – original value to encode

Returns encoded value

1.1.6 aries_cloudagent.ledger package

Subpackages

aries_cloudagent.ledger.tests package

Submodules

aries_cloudagent.ledger.tests.test_indy module

Submodules

aries_cloudagent.ledger.base module

Ledger base class.

class aries_cloudagent.ledger.base.**BaseLedger**
Bases: abc.ABC

Base class for ledger.

LEDGER_TYPE = None

did_to_nym(*did: str*) → str
Remove the ledger's DID prefix to produce a nym.

get_endpoint_for_did(*did: str*) → str
Fetch the endpoint for a ledger DID.

Parameters **did** – The DID to look up on the ledger or in the cache

get_key_for_did(*did: str*) → str
Fetch the verkey for a ledger DID.

Parameters **did** – The DID to look up on the ledger or in the cache

nym_to_did(*nym: str*) → str
Format a nym with the ledger's DID prefix.

update_endpoint_for_did(*did: str*, *endpoint: str*) → bool
Check and update the endpoint on the ledger.

Parameters

- **did** – The ledger DID
- **endpoint** – The endpoint address

aries_cloudagent.ledger.error module

Ledger related errors.

exception aries_cloudagent.ledger.error.**BadLedgerRequestError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.ledger.error.LedgerError*

The current request cannot proceed.

exception aries_cloudagent.ledger.error.**ClosedPoolError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.ledger.error.LedgerError*

Indy pool is closed.

exception aries_cloudagent.ledger.error.**LedgerConfigError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.ledger.error.LedgerError*

Base class for ledger configuration errors.

exception aries_cloudagent.ledger.error.**LedgerError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.error.BaseError*

Base class for ledger errors.

exception aries_cloudagent.ledger.error.**LedgerTransactionError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.ledger.error.LedgerError*

The ledger rejected the transaction.

aries_cloudagent.ledger.indy module

Indy ledger implementation.

class aries_cloudagent.ledger.indy.**IndyErrorHandler**(*message: str = None*, *error_cls: Type[aries_cloudagent.ledger.error.LedgerError] = <class 'aries_cloudagent.ledger.error.LedgerError'>*)

Bases: object

Trap IndyError and raise an appropriate LedgerError instead.

class aries_cloudagent.ledger.indy.**IndyLedger**(*pool_name: str*, *wallet: aries_cloudagent.wallet.base.BaseWallet*, ***, *keepalive: int = 0*, *cache: aries_cloudagent.cache.base.BaseCache = None*, *cache_duration: int = 600*)

Bases: *aries_cloudagent.ledger.base.BaseLedger*

Indy ledger class.

LEDGER_TYPE = 'indy'

TAA_ACCEPTED_RECORD_TYPE = 'taa_accepted'

accept_txn_author_agreement(*taa_record: dict*, *mechanism: str*, *accept_time: int = None*, *store: bool = False*)

Save a new record recording the acceptance of the TAA.

check_existing_schema(*public_did: str*, *schema_name: str*, *schema_version: str*, *attribute_names: Sequence[str]*) → str

Check if a schema has already been published.

check_pool_config() → bool

Check if a pool config has been created.

close()

Close the pool ledger.

create_pool_config(*genesis_transactions: str*, *recreate: bool = False*)

Create the pool ledger configuration.

fetch_credential_definition(*credential_definition_id: str*)

Get a credential definition from the ledger by id.

Parameters **credential_definition_id** – The schema id of the schema to fetch cred def for

fetch_schema(*schema_id: str*)

Get schema from ledger.

Parameters **schema_id** – The schema id to retrieve

fetch_txn_author_agreement()
Fetch the current AML and TAA from the ledger.

get_credential_definition(*credential_definition_id: str*)
Get a credential definition from the cache if available, otherwise the ledger.

Parameters **credential_definition_id** – The schema id of the schema to fetch cred def for

get_endpoint_for_did(*did: str*) → str
Fetch the endpoint for a ledger DID.

Parameters **did** – The DID to look up on the ledger or in the cache

get_indy_storage() → aries_cloudagent.storage.indy.IndyStorage
Get an IndyStorage instance for the current wallet.

get_key_for_did(*did: str*) → str
Fetch the verkey for a ledger DID.

Parameters **did** – The DID to look up on the ledger or in the cache

get_latest_txn_author_acceptance()
Look up the latest TAA acceptance.

get_schema(*schema_id: str*)
Get a schema from the cache if available, otherwise fetch from the ledger.

Parameters **schema_id** – The schema id to retrieve

get_txn_author_agreement(*reload: bool = False*)
Get the current transaction author agreement, fetching it if necessary.

nym_to_did(*nym: str*) → str
Format a nym with the ledger's DID prefix.

open()
Open the pool ledger, creating it if necessary.

send_credential_definition(*schema_id: str*, *tag: str = 'default'*)
Send credential definition to ledger and store relevant key matter in wallet.

Parameters

- **schema_id** – The schema id of the schema to create cred def for
- **tag** – Option tag to distinguish multiple credential definitions

send_schema(*schema_name: str, schema_version: str, attribute_names: Sequence[str]*)
Send schema to ledger.

Parameters

- **schema_name** – The schema name
- **schema_version** – The schema version
- **attribute_names** – A list of schema attributes

update_endpoint_for_did(*did: str*, *endpoint: str*) → bool
Check and update the endpoint on the ledger.

Parameters

- **did** – The ledger DID
- **endpoint** – The endpoint address

- **transport_vk** – The endpoint transport verkey

aries_cloudagent.ledger.provider module

Default ledger provider classes.

class aries_cloudagent.ledger.provider.**LedgerProvider**
Bases: *aries_cloudagent.config.base.BaseProvider*

Provider for the default ledger implementation.

LEDGER_CLASSES = {'indy': 'aries_cloudagent.ledger.indy.IndyLedger'}

provide(*settings: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Create and open the ledger instance.

1.1.7 aries_cloudagent.logging package

Subpackages

aries_cloudagent.logging.tests package

Submodules

aries_cloudagent.logging.tests.test_init module

1.1.8 aries_cloudagent.messaging package

Subpackages

aries_cloudagent.messaging.actionmenu package

Subpackages

aries_cloudagent.messaging.actionmenu.handlers package

Submodules

aries_cloudagent.messaging.actionmenu.handlers.menu_handler module

Action menu message handler.

class aries_cloudagent.messaging.actionmenu.handlers.menu_handler.**MenuHandler**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for action menus.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler logic for action menus.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.actionmenu.handlers.menu_request_handler module

Action menu request message handler.

class aries_cloudagent.messaging.actionmenu.handlers.menu_request_handler.**MenuRequestHandle**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for action menu requests.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler logic for action menu requests.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.actionmenu.handlers.perform_handler module

Action menu perform request message handler.

class aries_cloudagent.messaging.actionmenu.handlers.perform_handler.**PerformHandler**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for action menu perform requests.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler logic for action menu perform requests.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.actionmenu.messages package

Subpackages

aries_cloudagent.messaging.actionmenu.messages.tests package

Submodules

aries_cloudagent.messaging.actionmenu.messages.tests.perform_test module

class aries_cloudagent.messaging.actionmenu.messages.tests.perform_test.**TestPerform**(*methodNa*
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_perform_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_make_model()

test_name = 'option_name'

test_params = {'a': 'aaa'}

test_serialize(*mock_perform_schema_dump*)
Test serialization.

test_type()
Test type.

aries_cloudagent.messaging.actionmenu.messages.tests.test_menu module

class aries_cloudagent.messaging.actionmenu.messages.tests.test_menu.**TestConfig**
Bases: object

test_menu_message = {'description': 'IIWBook facilitates connections between attendee

class aries_cloudagent.messaging.actionmenu.messages.tests.test_menu.**TestMenu**(*methodName='ru*
Bases: unittest.case.TestCase, *aries_cloudagent.messaging.actionmenu.messages.tests.test_menu.TestConfig*

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_menu_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_make_model()

test_serialize(*mock_menu_schema_dump*)
Test serialization.

test_type()
Test type.

aries_cloudagent.messaging.actionmenu.messages.tests.test_menu_request module

class aries_cloudagent.messaging.actionmenu.messages.tests.test_menu_request.**TestMenuReques**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_menu_request_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_make_model()

test_serialize(*mock_menu_request_schema_dump*)
Test serialization.

test_type()
Test type.

Submodules

aries_cloudagent.messaging.actionmenu.messages.menu module

Represents an action menu.

class aries_cloudagent.messaging.actionmenu.messages.menu.**Menu**(*, *title: str = None*, *description: str = None*, *errormsg: str = None*, *options: Sequence[aries_cloudagent.messaging.acti = None*, ***kwargs*)
Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing an action menu.

class Meta
Bases: object

Metadata for an action menu.

handler_class = 'aries_cloudagent.messaging.actionmenu.handlers.menu_handler.MenuHa

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/action-menu/1.0/menu'

schema_class = 'MenuSchema'

class aries_cloudagent.messaging.actionmenu.messages.menu.**MenuSchema**(**args*, ***kwargs*)
Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Menu schema class.

class Meta
Bases: object

Menu schema metadata.

model_class
alias of *Menu*

description
Used by autodoc_mock_imports.

errormsg
Used by autodoc_mock_imports.

options
Used by autodoc_mock_imports.

title
Used by autodoc_mock_imports.

aries_cloudagent.messaging.actionmenu.messages.menu_request module

Represents a request for an action menu.

class aries_cloudagent.messaging.actionmenu.messages.menu_request.**MenuRequest**(**kwargs)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a request for an action menu.

class Meta

Bases: object

Metadata for action menu request.

handler_class = 'aries_cloudagent.messaging.actionmenu.handlers.menu_request_handle

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/action-menu/1.0/menu-request'

schema_class = 'MenuRequestSchema'

class aries_cloudagent.messaging.actionmenu.messages.menu_request.**MenuRequestSchema**(*args, **kwargs

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

MenuRequest schema class.

class Meta

Bases: object

MenuRequest schema metadata.

model_class

alias of *MenuRequest*

aries_cloudagent.messaging.actionmenu.messages.perform module

Represents a request to perform a menu action.

class aries_cloudagent.messaging.actionmenu.messages.perform.**Perform**(*, *name: str = None*, *params: Mapping[str, str] = None*, **kwargs)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a request to perform a menu action.

class Meta

Bases: object

Perform metadata.

handler_class = 'aries_cloudagent.messaging.actionmenu.handlers.perform_handler.Per

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/action-menu/1.0/perform'

schema_class = 'PerformSchema'

class aries_cloudagent.messaging.actionmenu.messages.perform.**PerformSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Perform schema class.

class Meta

Bases: object

Perform schema metadata.

model_class

alias of *Perform*

name

Used by autodoc_mock_imports.

params

Used by autodoc_mock_imports.

aries_cloudagent.messaging.actionmenu.models package

Submodules

aries_cloudagent.messaging.actionmenu.models.menu_form module

Record used to represent the form associated with an action menu option.

class aries_cloudagent.messaging.actionmenu.models.menu_form.**MenuForm**(**, title: str = None, description: str = None, params: Sequence[aries_cloudagent.mess = None, submit_label: str = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Instance of a form associated with an action menu item.

class Meta

Bases: object

Menu form metadata.

schema_class = 'MenuFormSchema'

class aries_cloudagent.messaging.actionmenu.models.menu_form.**MenuFormSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

MenuForm schema.

class Meta

Bases: object

MenuFormSchema metadata.

model_class

alias of *MenuForm*

description

Used by autodoc_mock_imports.

params

Used by autodoc_mock_imports.

submit_label

Used by autodoc_mock_imports.

title

Used by autodoc_mock_imports.

aries_cloudagent.messaging.actionmenu.models.menu_form_param module

Record used to represent a parameter in a menu form.

class aries_cloudagent.messaging.actionmenu.models.menu_form_param.**MenuFormParam**(*, *name: str = None*, *title: str = None*, *default: str = None*, *description: str = None*, *input_type: str = None*, *required: bool = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Instance of a menu form param associated with an action menu option.

class Meta

Bases: object

Menu form param metadata.

schema_class = 'MenuFormParamSchema'

class aries_cloudagent.messaging.actionmenu.models.menu_form_param.**MenuFormParamSchema**(**args*, ***kw*

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

MenuFormParam schema.

class Meta

Bases: object

MenuFormParamSchema metadata.

model_class

alias of *MenuFormParam*

default

Used by autodoc_mock_imports.

description
Used by autodoc_mock_imports.

input_type
Used by autodoc_mock_imports.

name
Used by autodoc_mock_imports.

required
Used by autodoc_mock_imports.

title
Used by autodoc_mock_imports.

aries_cloudagent.messaging.actionmenu.models.menu_option module

Record used to represent individual menu options in an action menu.

class aries_cloudagent.messaging.actionmenu.models.menu_option.**MenuOption**(*, *name: str = None*, *title: str = None*, *description: str = None*, *disabled: bool = None*, *form: aries_cloudagent.messag = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Instance of a menu option associated with an action menu.

class Meta
Bases: object

Menu option metadata.

schema_class = 'MenuOptionSchema'

class aries_cloudagent.messaging.actionmenu.models.menu_option.**MenuOptionSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

MenuOption schema.

class Meta
Bases: object

MenuOptionSchema metadata.

model_class
alias of *MenuOption*

description
Used by autodoc_mock_imports.

disabled
Used by autodoc_mock_imports.

form
Used by autodoc_mock_imports.

name
Used by autodoc_mock_imports.

title
Used by autodoc_mock_imports.

Submodules

aries_cloudagent.messaging.actionmenu.base_service module

Base action menu service classes.

class aries_cloudagent.messaging.actionmenu.base_service.**BaseMenuService**(*context: aries_cloudagent.config.ir*

Bases: abc.ABC

Base action menu service interface.

get_active_menu(*connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord = None*, *thread_id: str = None*) → aries_cloudagent.messaging.actionmenu.messages.menu.Menu
Render the current menu.

Parameters

- **connection** – The active connection record
- **thread_id** – The thread identifier from the requesting message.

perform_menu_action(*action_name: str*, *action_params: dict*, *connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord = None*, *thread_id: str = None*) → aries_cloudagent.messaging.agent_message.AgentMessage
Perform an action defined by the active menu.

Parameters

- **action_name** – The unique name of the action being performed
- **action_params** – A collection of parameters for the action
- **connection** – The active connection record
- **thread_id** – The thread identifier from the requesting message.

classmethod service_handler()
Quick accessor for conductor to use.

aries_cloudagent.messaging.actionmenu.controller module

Protocol controller for the action menu message family.

class aries_cloudagent.messaging.actionmenu.controller.**Controller**(*protocol: str*)
Bases: object

Action menu protocol controller.

determine_roles(*context: aries_cloudagent.config.injection_context.InjectionContext*) → Sequence[str]
Determine what action menu roles are defined.

aries_cloudagent.messaging.actionmenu.driver_service module

Driver-based action menu service classes.

class aries_cloudagent.messaging.actionmenu.driver_service.**DriverMenuService**(*context: aries_cloudagent.co*
Bases: *aries_cloudagent.messaging.actionmenu.base_service.BaseMenuService*

Driver-based action menu service.

get_active_menu(*connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord = None*, *thread_id: str = None*) → aries_cloudagent.messaging.actionmenu.messages.menu.Menu
Render the current menu.

Parameters

- **connection** – The active connection record
- **thread_id** – The thread identifier from the requesting message.

perform_menu_action(*action_name: str*, *action_params: dict*, *connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord = None*, *thread_id: str = None*) → aries_cloudagent.messaging.agent_message.AgentMessage
Perform an action defined by the active menu.

Parameters

- **action_name** – The unique name of the action being performed
- **action_params** – A collection of parameters for the action
- **connection** – The active connection record
- **thread_id** – The thread identifier from the requesting message.

send_webhook(*topic: str*, *payload: dict*)
Dispatch a webhook through the registered responder.

aries_cloudagent.messaging.actionmenu.message_types module

Message type identifiers for Action Menus.

aries_cloudagent.messaging.actionmenu.routes module

Action menu admin routes.

class aries_cloudagent.messaging.actionmenu.routes.**MenuJsonSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Matches MenuSchema but without the inherited AgentMessage properties.

description
Used by autodoc_mock_imports.

errormsg
Used by autodoc_mock_imports.

options
Used by autodoc_mock_imports.

title
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.actionmenu.routes.**PerformRequestSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for performing a menu action.

name
Used by autodoc_mock_imports.

params
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.actionmenu.routes.**SendMenuSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for sending a menu to a connection.

menu
Used by autodoc_mock_imports.

aries_cloudagent.messaging.actionmenu.routes.**actionmenu_close**(*request: <sphinx.ext.autodoc.importer._MockObjec object at 0x7fba4a04e278>*)

Request handler for closing the menu associated with a connection.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.actionmenu.routes.**actionmenu_fetch**(*request: <sphinx.ext.autodoc.importer._MockObjec object at 0x7fba4a04e278>*)

Request handler for fetching the previously-received menu for a connection.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.actionmenu.routes.**actionmenu_perform**(*request: <sphinx.ext.autodoc.importer._MockOb object at 0x7fba4a04e278>*)

Request handler for performing a menu action.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.actionmenu.routes.**actionmenu_request**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a04e278>*)

Request handler for requesting a menu from the connection target.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.actionmenu.routes.**actionmenu_send**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a04e278>*)

Request handler for requesting a menu from the connection target.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.actionmenu.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a04e278>*)

Register routes.

aries_cloudagent.messaging.actionmenu.util module

Action menu utility methods.

aries_cloudagent.messaging.actionmenu.util.**retrieve_connection_menu**(*connection_id: str, context: aries_cloudagent.config.injection → aries_cloudagent.messaging.actio*

Retrieve the previously-received action menu.

aries_cloudagent.messaging.actionmenu.util.**save_connection_menu**(*menu: aries_cloudagent.messaging.actionmen connection_id: str, context: aries_cloudagent.config.injection_conte*

Save a received action menu.

aries_cloudagent.messaging.basicmessage package

Subpackages

aries_cloudagent.messaging.basicmessage.handlers package

Submodules

aries_cloudagent.messaging.basicmessage.handlers.basicmessage_handler module

Basic message handler.

class aries_cloudagent.messaging.basicmessage.handlers.basicmessage_handler.**BasicMessageHa**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for basic messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler logic for basic messages.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.basicmessage.messages package

Subpackages

aries_cloudagent.messaging.basicmessage.messages.tests package

Submodules

aries_cloudagent.messaging.basicmessage.messages.tests.test_basic_message module

class aries_cloudagent.messaging.basicmessage.messages.tests.test_basic_message.**TestBasicMe**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_basic_message_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_serialize(*mock_basic_message_schema_load*)
Test serialization.

test_type()
Test type.

class aries_cloudagent.messaging.basicmessage.messages.tests.test_basic_message.**TestBasicMe**

Bases: sphinx.ext.autodoc.importer._MockObject

Test basic message schema.

test_make_model()

Submodules

aries_cloudagent.messaging.basicmessage.messages.basicmessage module

Basic message.

class aries_cloudagent.messaging.basicmessage.messages.basicmessage.**BasicMessage**(*, *sent_time: Union[str, datetime.datetime] = None*, *content: str = None*, *localization: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class defining the structure of a basic message.

class Meta

Bases: object

Basic message metadata class.

handler_class = 'aries_cloudagent.messaging.basicmessage.handlers.basicmessage_hand

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/basicmessage/1.0/message'

schema_class = 'BasicMessageSchema'

class aries_cloudagent.messaging.basicmessage.messages.basicmessage.**BasicMessageSchema**(**args*, ***kw*

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Basic message schema class.

class Meta

Bases: object

Basic message schema metadata.

model_class

alias of *BasicMessage*

content

Used by autodoc_mock_imports.

localization

Used by autodoc_mock_imports.

sent_time

Used by autodoc_mock_imports.

Submodules

aries_cloudagent.messaging.basicmessage.message_types module

Message type identifiers for Connections.

aries_cloudagent.messaging.basicmessage.routes module

Basic message admin routes.

class aries_cloudagent.messaging.basicmessage.routes.**SendMessageSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for sending a message.

content

Used by autodoc_mock_imports.

aries_cloudagent.messaging.basicmessage.routes.**connections_expire_message**(*request: <sphinx.ext.autodoc.imp*)

Request handler for sending a basic message to a connection.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.basicmessage.routes.**connections_send_message**(*request: <sphinx.ext.autodoc.import*)

Request handler for sending a basic message to a connection.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.basicmessage.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f8fdd8>*)

Register routes.

aries_cloudagent.messaging.connections package

Subpackages

aries_cloudagent.messaging.connections.handlers package

Submodules

aries_cloudagent.messaging.connections.handlers.connection_invitation_handler module

Connect invitation handler.

class aries_cloudagent.messaging.connections.handlers.connection_invitation_handler.**Connect**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler class for connection invitations.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Handle connection invitation.

Parameters

- **context** – Request context
- **responder** – Responder callback

aries_cloudagent.messaging.connections.handlers.connection_request_handler module

Connection request handler.

class aries_cloudagent.messaging.connections.handlers.connection_request_handler.**Connectio**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler class for connection requests.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Handle connection request.

Parameters

- **context** – Request context
- **responder** – Responder callback

aries_cloudagent.messaging.connections.handlers.connection_response_handler module

Connection response handler.

class aries_cloudagent.messaging.connections.handlers.connection_response_handler.**Connectio**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler class for connection responses.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Handle connection response.

Parameters

- **context** – Request context
- **responder** – Responder callback

aries_cloudagent.messaging.connections.messages package

Subpackages

aries_cloudagent.messaging.connections.messages.tests package

Submodules

aries_cloudagent.messaging.connections.messages.tests.test_connection_invitation module

class aries_cloudagent.messaging.connections.messages.tests.test_connection_invitation.**Test**
Bases: unittest.case.TestCase

did = 'did:sov:QmWbsNYhMrjHiqZDTUTEJs'

endpoint_did = 'did:sov:A2wBhNYhMrjHiqZDTUYH7u'

endpoint_url = 'https://example.com/endpoint'

image_url = 'https://example.com/image.jpg'

key = '8HH5gYEeNc3z7PYXmd54d4x6qAfCNrqQqEB3nS7Zfu7K'

label = 'Label'

test_deserialize(*mock_connection_invitation_schema_load*)

test_init()

test_serialize(*mock_connection_invitation_schema_dump*)

test_type()

test_url_round_trip()

class aries_cloudagent.messaging.connections.messages.tests.test_connection_invitation.**Test**
Bases: unittest.case.TestCase

connection_invitation = <ConnectionInvitation(_message_id='3d7ced15-9c4e-444d-ab9f-035

test_make_model()

aries_cloudagent.messaging.connections.messages.tests.test_connection_request module

class aries_cloudagent.messaging.connections.messages.tests.test_connection_request.**TestCo**
Bases: object

make_did_doc()

test_did = '55GkHamhTU1ZbTbV2ab9DE'

test_endpoint = 'http://localhost'

test_label = 'Label'

test_seed = 'testseed000000000000000000000001'

test_verkey = '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'

class aries_cloudagent.messaging.connections.messages.tests.test_connection_request.**TestCo**
Bases: unittest.case.TestCase, *aries_cloudagent.messaging.connections.messages.tests.test_connection_request.TestConfig*

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_connection_request_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_serialize(*mock_connection_request_schema_dump*)
Test serialization.

test_type()
Test type.

class aries_cloudagent.messaging.connections.messages.tests.test_connection_request.**TestCon**

Bases: sphinx.ext.autodoc.importer._MockObject, *aries_cloudagent.messaging.connections.messages.tests.test_connection_request.TestConfig*

Test connection request schema.

test_make_model()

aries_cloudagent.messaging.connections.messages.tests.test_connection_response module

class aries_cloudagent.messaging.connections.messages.tests.test_connection_response.**TestCo**
Bases: object

make_did_doc()

test_did = '55GkHamhTU1ZbTbV2ab9DE'

test_endpoint = 'http://localhost'

test_seed = 'testseed000000000000000000000001'

test_verkey = '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'

class aries_cloudagent.messaging.connections.messages.tests.test_connection_response.**TestCo**
Bases: unittest.case.TestCase, *aries_cloudagent.messaging.connections.messages.tests.test_connection_response.TestConfig*

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_connection_response_schema_load*)
Test deserialization.

test_init()

test_serialize(*mock_connection_response_schema_dump*)
Test serialization.

test_type()

class aries_cloudagent.messaging.connections.messages.tests.test_connection_response.**TestCo**

Bases: sphinx.ext.autodoc.importer._MockObject, *aries_cloudagent.messaging.connections.messages.tests.test_connection_response.TestConfig*

test_make_model()

Submodules

aries_cloudagent.messaging.connections.messages.connection_invitation module

Represents an invitation message for establishing connection.

class aries_cloudagent.messaging.connections.messages.connection_invitation.**ConnectionInvit**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a connection invitation.

class Meta

Bases: object

Metadata for a connection invitation.

handler_class = 'aries_cloudagent.messaging.connections.handlers.connection_invitat

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/invitation'

schema_class = 'ConnectionInvitationSchema'

classmethod from_url(*url: str*) → aries_cloudagent.messaging.connections.messages.connection_invitation.ConnectionI

Parse a URL-encoded invitation into a *ConnectionInvitation* message.

Parameters **url** – Url to decode

Returns A *ConnectionInvitation* object.

to_url() → str
Convert an invitation to URL format for sharing.

Returns An invite url

class aries_cloudagent.messaging.connections.messages.connection_invitation.**ConnectionInvit**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Connection invitation schema class.

class Meta
Bases: object

Connection invitation schema metadata.

model_class
alias of *ConnectionInvitation*

did
Used by autodoc_mock_imports.

endpoint
Used by autodoc_mock_imports.

image_url
Used by autodoc_mock_imports.

label
Used by autodoc_mock_imports.

recipient_keys
Used by autodoc_mock_imports.

routing_keys
Used by autodoc_mock_imports.

validate_fields(*data*)
Validate schema fields.

Parameters **data** – The data to validate

Raises ValidationError – If any of the fields do not validate

aries_cloudagent.messaging.connections.messages.connection_request module

Represents a connection request message.

class aries_cloudagent.messaging.connections.messages.connection_request.**ConnectionRequest**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a connection request.

class Meta

Bases: object

Metadata for a connection request.

handler_class = 'aries_cloudagent.messaging.connections.handlers.connection_request

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/request'

schema_class = 'ConnectionRequestSchema'

class aries_cloudagent.messaging.connections.messages.connection_request.**ConnectionRequestS**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Connection request schema class.

class Meta

Bases: object

Connection request schema metadata.

model_class

alias of *ConnectionRequest*

connection

Used by autodoc_mock_imports.

image_url

Used by autodoc_mock_imports.

label

Used by autodoc_mock_imports.

aries_cloudagent.messaging.connections.messages.connection_response module

Represents a connection response message.

class aries_cloudagent.messaging.connections.messages.connection_response.**ConnectionRespons**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a connection response.

class Meta

Bases: object

Metadata for a connection response.

handler_class = 'aries_cloudagent.messaging.connections.handlers.connection_respons

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/response'

schema_class = 'ConnectionResponseSchema'

class aries_cloudagent.messaging.connections.messages.connection_response.**ConnectionRespons**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Connection response schema class.

class Meta

Bases: object

Connection response schema metadata.

model_class

alias of *ConnectionResponse*

signed_fields = ('connection',)

connection

Used by autodoc_mock_imports.

aries_cloudagent.messaging.connections.messages.problem_report module

Represents a connection problem report message.

class aries_cloudagent.messaging.connections.messages.problem_report.**ProblemReport**(*, *problem_code: str = None*, *explain: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Base class representing a connection problem report message.

class Meta
Bases: object

Connection problem report metadata.

handler_class = 'aries_cloudagent.messaging.problem_report.handler.ProblemReportHar

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/problem_report'

schema_class = 'ProblemReportSchema'

class aries_cloudagent.messaging.connections.messages.problem_report.**ProblemReportReason**
Bases: str, enum.Enum

Supported reason codes.

INVITATION_NOT_ACCEPTED = 'invitation_not_accepted'

REQUEST_NOT_ACCEPTED = 'request_not_accepted'

REQUEST_PROCESSING_ERROR = 'request_processing_error'

RESPONSE_NOT_ACCEPTED = 'response_not_accepted'

RESPONSE_PROCESSING_ERROR = 'response_processing_error'

class aries_cloudagent.messaging.connections.messages.problem_report.**ProblemReportSchema**(*a
**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Schema for ProblemReport base class.

class Meta
Bases: object

Metadata for problem report schema.

model_class
alias of *ProblemReport*

explain
Used by autodoc_mock_imports.

problem_code
Used by autodoc_mock_imports.

aries_cloudagent.messaging.connections.models package

Subpackages

aries_cloudagent.messaging.connections.models.diddoc package

DID Document model support.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class aries_cloudagent.messaging.connections.models.diddoc.**DIDDoc**(*did: str = None*)

Bases: object

DID document, grouping a DID with verification keys and services.

Retains DIDs as raw values (orientated toward indy-facing operations), everything else as URIs (oriented toward W3C-facing operations).

CONTEXT = 'https://w3id.org/did/v1'

add_service_pubkeys(*service: dict, tags: Union[Sequence[str], str]*) → List[aries_cloudagent.messaging.connections.models.diddoc.publickey.PublicKey]

Add public keys specified in service. Return public keys so discovered.

Parameters

- **service** – service from DID document
- **tags** – potential tags marking public keys of type of interest (the standard is still coalescing)

Raises ValueError – for public key reference not present in DID document.

Returns: list of public keys from the document service specification

authnkey

Accessor for public keys marked as authentication keys, by identifier.

classmethod deserialize(*did_doc: dict*) → aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc

Construct DIDDoc object from dict representation.

Parameters **did_doc** – DIDDoc dict representation

Raises ValueError – for bad DID or missing mandatory item.

Returns: DIDDoc from input json

did

Accessor for DID.

classmethod from_json(*did_doc_json: str*) → aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc

Construct DIDDoc object from json representation.

Parameters **did_doc_json** – DIDDoc json representation

Returns: DIDDoc from input json

pubkey

Accessor for public keys by identifier.

serialize() → str

Dump current object to a JSON-compatible dictionary.

Returns dict representation of current DIDDoc

service

Accessor for services by identifier.

set(*item: Union[aries_cloudagent.messaging.connections.models.diddoc.service.Service, aries_cloudagent.messaging.connections.models.diddoc.publickey.PublicKey]*) → aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc
Add or replace service or public key; return current DIDDoc.

Raises ValueError – if input item is neither service nor public key.

Parameters **item** – service or public key to set

Returns: the current DIDDoc

to_json() → str
Dump current object as json (JSON-LD).

Returns json representation of current DIDDoc

class aries_cloudagent.messaging.connections.models.diddoc.**LinkedDataKeySpec**(*ver_type*, *authn_type*, *specifier*)

Bases: tuple

authn_type
Alias for field number 1

specifier
Alias for field number 2

ver_type
Alias for field number 0

class aries_cloudagent.messaging.connections.models.diddoc.**PublicKey**(*did: str*, *ident: str*, *value: str*, *pk_type: aries_cloudagent.messaging.co* = *None*, *controller: str* = *None*, *authn: bool* = *False*)

Bases: object

Public key specification to embed in DID document.

Retains DIDs as raw values (orientated toward indy-facing operations), everything else as URIs (oriented toward W3C-facing operations).

authn
Accessor for the authentication marker.

Returns: whether public key is marked as having DID authentication privilege

controller
Accessor for the controller DID.

did
Accessor for the DID.

id
Accessor for the public key identifier.

to_dict() → dict
Return dict representation of public key to embed in DID document.

type
Accessor for the public key type.

value
Accessor for the public key value.

class aries_cloudagent.messaging.connections.models.diddoc.**PublicKeyType**
Bases: enum.Enum

Class encapsulating public key types.

ED25519_SIG_2018 = LinkedDataKeySpec(ver_type='Ed25519VerificationKey2018', authn_type=

EDDSA_SA_SIG_SECP256K1 = LinkedDataKeySpec(ver_type='Secp256k1VerificationKey2018', au

RSA_SIG_2018 = LinkedDataKeySpec(ver_type='RsaVerificationKey2018', authn_type='RsaSig

authn_type
Accessor for the authentication type identifier.

get = <function PublicKeyType.get>

specification(*val: str*) → str
Return specifier and input value for use in public key specification.

Parameters **val** – value of public key

Returns: dict mapping applicable specifier to input value

specifier
Accessor for the value specifier.

ver_type
Accessor for the verification type identifier.

class aries_cloudagent.messaging.connections.models.diddoc.**Service**(*did: str, ident: str, typ: str, recip_keys: Union[Sequence[T_co], aries_cloudagent.messaging.conne routing_keys: Union[Sequence[T_co], aries_cloudagent.messaging.conne endpoint: str, priority: int = 0*)

Bases: object

Service specification to embed in DID document.

Retains DIDs as raw values (orientated toward indy-facing operations), everything else as URIs (oriented toward W3C-facing operations).

did
Accessor for the DID value.

endpoint
Accessor for the endpoint value.

id
Accessor for the service identifier.

priority
Accessor for the priority value.

recip_keys
Accessor for the recipient keys.

routing_keys
Accessor for the routing keys.

to_dict() → dict
Return dict representation of service to embed in DID document.

type
Accessor for the service type.

Submodules

aries_cloudagent.messaging.connections.models.diddoc.diddoc module

DID Document classes.

class aries_cloudagent.messaging.connections.models.diddoc.diddoc.**DIDDoc**(*did: str = None*)

Bases: object

DID document, grouping a DID with verification keys and services.

Retains DIDs as raw values (orientated toward indy-facing operations), everything else as URIs (oriented toward W3C-facing operations).

CONTEXT = 'https://w3id.org/did/v1'

add_service_pubkeys(*service: dict, tags: Union[Sequence[str], str]*) → List[aries_cloudagent.messaging.connections.models.diddoc.publickey.PublicKey]
Add public keys specified in service. Return public keys so discovered.

Parameters

- **service** – service from DID document
- **tags** – potential tags marking public keys of type of interest (the standard is still coalescing)

Raises ValueError – for public key reference not present in DID document.

Returns: list of public keys from the document service specification

authnkey
Accessor for public keys marked as authentication keys, by identifier.

classmethod deserialize(*did_doc: dict*) → aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc
Construct DIDDoc object from dict representation.

Parameters **did_doc** – DIDDoc dict representation

Raises ValueError – for bad DID or missing mandatory item.

Returns: DIDDoc from input json

did
Accessor for DID.

classmethod from_json(*did_doc_json: str*) → aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc
Construct DIDDoc object from json representation.

Parameters **did_doc_json** – DIDDoc json representation

Returns: DIDDoc from input json

pubkey
Accessor for public keys by identifier.

serialize() → str
Dump current object to a JSON-compatible dictionary.

Returns dict representation of current DIDDoc

service
Accessor for services by identifier.

set(*item: Union[aries_cloudagent.messaging.connections.models.diddoc.service.Service, aries_cloudagent.messaging.connections.models.diddoc.publickey.PublicKey]*) → aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc
Add or replace service or public key; return current DIDDoc.

Raises ValueError – if input item is neither service nor public key.

Parameters **item** – service or public key to set

Returns: the current DIDDoc

to_json() → str
Dump current object as json (JSON-LD).

Returns json representation of current DIDDoc

aries_cloudagent.messaging.connections.models.diddoc.publickey module

DID Document Public Key classes.

class aries_cloudagent.messaging.connections.models.diddoc.publickey.**LinkedDataKeySpec**(*ver_t au- thn_t spec- i- fier*)

Bases: tuple

authn_type
Alias for field number 1

specifier
Alias for field number 2

ver_type
Alias for field number 0

class aries_cloudagent.messaging.connections.models.diddoc.publickey.**PublicKey**(*did: str, ident: str, value: str, pk_type: aries_cloudagent = None, con- troller: str = None, au- thn: bool = False*)

Bases: object

Public key specification to embed in DID document.

Retains DIDs as raw values (orientated toward indy-facing operations), everything else as URIs (oriented toward W3C-facing operations).

authn
Accessor for the authentication marker.

Returns: whether public key is marked as having DID authentication privilege

controller
Accessor for the controller DID.

did
Accessor for the DID.

id
Accessor for the public key identifier.

to_dict() → dict
Return dict representation of public key to embed in DID document.

type
Accessor for the public key type.

value
Accessor for the public key value.

class aries_cloudagent.messaging.connections.models.diddoc.publickey.**PublicKeyType**
Bases: enum.Enum

Class encapsulating public key types.

ED25519_SIG_2018 = LinkedDataKeySpec(ver_type='Ed25519VerificationKey2018', authn_type

EDDSA_SA_SIG_SECP256K1 = LinkedDataKeySpec(ver_type='Secp256k1VerificationKey2018', au

RSA_SIG_2018 = LinkedDataKeySpec(ver_type='RsaVerificationKey2018', authn_type='RsaSig

authn_type
Accessor for the authentication type identifier.

get = <function PublicKeyType.get>

specification(*val: str*) → str
Return specifier and input value for use in public key specification.

Parameters **val** – value of public key

Returns: dict mapping applicable specifier to input value

specifier
Accessor for the value specifier.

ver_type
Accessor for the verification type identifier.

aries_cloudagent.messaging.connections.models.diddoc.service module

DID Document Service classes.

class aries_cloudagent.messaging.connections.models.diddoc.service.**Service**(*did: str, ident: str, typ: str, recip_keys: Union[Sequence[T_co, aries_cloudagent.mess routing_keys: Union[Sequence[T_co, aries_cloudagent.mess endpoint: str, priority: int = 0*)

Bases: object

Service specification to embed in DID document.

Retains DIDs as raw values (orientated toward indy-facing operations), everything else as URIs (oriented toward W3C-facing operations).

did
Accessor for the DID value.

endpoint
Accessor for the endpoint value.

id
Accessor for the service identifier.

priority
Accessor for the priority value.

recip_keys
Accessor for the recipient keys.

routing_keys
Accessor for the routing keys.

to_dict() → dict
Return dict representation of service to embed in DID document.

type
Accessor for the service type.

aries_cloudagent.messaging.connections.models.diddoc.util module

DIDDoc utility methods.

aries_cloudagent.messaging.connections.models.diddoc.util.**canon_did**(*uri: str*) → str

Convert a URI into a DID if need be, left-stripping 'did:sov:' if present.

Parameters **uri** – input URI or DID

Raises ValueError – for invalid input.

aries_cloudagent.messaging.connections.models.diddoc.util.**canon_ref**(*did: str*, *ref: str*, *delimiter: str = None*)

Given a reference in a DID document, return it in its canonical form of a URI.

Parameters

- **did** – DID acting as the identifier of the DID document
- **ref** – reference to canonicalize, either a DID or a fragment pointing to a location in the DID doc
- **delimiter** – delimiter character marking fragment (default '#') or introducing identifier (';') against DID resource

aries_cloudagent.messaging.connections.models.diddoc.util.**ok_did**(*token: str*) → bool

Whether input token looks like a valid distributed identifier.

Parameters **token** – candidate string

Returns: whether input token looks like a valid schema identifier

aries_cloudagent.messaging.connections.models.diddoc.util.**resource**(*ref: str*, *delimiter: str = None*) → str

Extract the resource for an identifier.

Given a (URI) reference, return up to its delimiter (exclusively), or all of it if there is none.

Parameters

- **ref** – reference
- **delimiter** – delimiter character (default None maps to '#', or ';' introduces identifiers)

Submodules

aries_cloudagent.messaging.connections.models.connection_detail module

An object for containing the connection request/response DID information.

class aries_cloudagent.messaging.connections.models.connection_detail.**ConnectionDetail**(*, *did: str = None*, *did_doc = None*, *aries = None*, ***kw*

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing the details of a connection.

class Meta

Bases: object

ConnectionDetail metadata.

schema_class = 'ConnectionDetailSchema'

did

Accessor for the connection DID.

Returns The DID for this connection

did_doc

Accessor for the connection DID Document.

Returns The DIDDoc for this connection

class aries_cloudagent.messaging.connections.models.connection_detail.**ConnectionDetailSche**

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

ConnectionDetail schema.

class Meta

Bases: object

ConnectionDetailSchema metadata.

model_class = 'ConnectionDetail'

did

Used by autodoc_mock_imports.

did_doc

Field that loads and serializes DIDDoc.

class aries_cloudagent.messaging.connections.models.connection_detail.**DIDDocWrapper**(**args*, ***kwargs*

Bases: sphinx.ext.autodoc.importer._MockObject

Field that loads and serializes DIDDoc.

aries_cloudagent.messaging.connections.models.connection_record module

Handle connection information interface with non-secrets storage.

class aries_cloudagent.messaging.connections.models.connection_record.**ConnectionRecord**(*, *connection_* *str* = *None* *my_d* *str* = *None* *their_* *str* = *None* *their_* *str* = *None* *their_* *str* = *None* *initiator:* *str* = *None* *invitation_* *str* = *None* *request* *str* = *None* *state:* *str* = *None* *inboun* *str* = *None* *error_r* *str* = *None* *routing_s* *str* = *None*

Bases: aries_cloudagent.messaging.models.base_record.BaseRecord

Represents a single pairwise connection.

ACCEPT_AUTO = 'auto'

ACCEPT_MANUAL = 'manual'

CACHE_ENABLED = True

DIRECTION_RECEIVED = 'received'

DIRECTION_SENT = 'sent'

INITIATOR_EXTERNAL = 'external'

INITIATOR_SELF = 'self'

LOG_STATE_FLAG = 'debug.connections'

class Meta

Bases: object

ConnectionRecord metadata.

schema_class = 'ConnectionRecordSchema'

RECORD_ID_NAME = 'connection_id'

RECORD_TYPE = 'connection'

RECORD_TYPE_ACTIVITY = 'connection_activity'

RECORD_TYPE_INVITATION = 'connection_invitation'

RECORD_TYPE_REQUEST = 'connection_request'

ROUTING_STATE_ACTIVE = 'active'

ROUTING_STATE_ERROR = 'error'

ROUTING_STATE_NONE = 'none'

ROUTING_STATE_REQUEST = 'request'

STATE_ACTIVE = 'active'

STATE_ERROR = 'error'

STATE_INACTIVE = 'inactive'

STATE_INIT = 'init'

STATE_INVITATION = 'invitation'

STATE_REQUEST = 'request'

STATE_RESPONSE = 'response'

WEBHOOK_TOPIC = 'connections'

WEBHOOK_TOPIC_ACTIVITY = 'connections_activity'

attach_invitation(*context: aries_cloudagent.config.injection_context.InjectionContext*, *invitation: aries_cloudagent.messaging.connections.messages.connection_invitation.ConnectionInvitation*)

Persist the related connection invitation to storage.

Parameters

- **context** – The injection context to use

- **invitation** – The invitation to relate to this connection record

attach_request(*context: aries_cloudagent.config.injection_context.InjectionContext*, *request: aries_cloudagent.messaging.connections.messages.connection_request.ConnectionRequest*)
Persist the related connection request to storage.

Parameters

- **context** – The injection context to use
- **request** – The request to relate to this connection record

connection_id
Accessor for the ID associated with this connection.

fetch_activity(*context: aries_cloudagent.config.injection_context.InjectionContext*, *activity_type: str = None*, *direction: str = None*) → Sequence[dict]
Fetch all activity logs for this connection record.

Parameters

- **context** – The injection context to use
- **activity_type** – An optional activity type filter
- **direction** – An optional direction filter

is_ready
Accessor for connection readiness.

log_activity(*context: aries_cloudagent.config.injection_context.InjectionContext*, *activity_type: str*, *direction: str*, *meta: dict = None*)
Log an event against this connection record.

Parameters

- **context** – The injection context to use
- **activity_type** – The activity type identifier
- **direction** – The direction of the activity (sent or received)
- **meta** – Optional metadata for the activity

post_save(*context: aries_cloudagent.config.injection_context.InjectionContext*, **args*, ***kwargs*)
Perform post-save actions.

Parameters **context** – The injection context to use

record_tags
Accessor for the record tags generated for this connection.

record_value
Accessor to for the JSON record value properties for this connection.

retrieve_activity(*context: aries_cloudagent.config.injection_context.InjectionContext*, *activity_id: str*) → Sequence[dict]
Retrieve a single activity record.

Parameters

- **context** – The injection context to use
- **activity_id** – The ID of the activity entry

classmethod retrieve_by_did(*context: aries_cloudagent.config.injection_context.InjectionContext*, *their_did: str = None*, *my_did: str = None*, *initiator: str = None*) → aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord

Retrieve a connection record by target DID.

Parameters

- **context** – The injection context to use
- **their_did** – The target DID to filter by
- **my_did** – One of our DIDs to filter by
- **initiator** – Filter connections by the initiator value

classmethod retrieve_by_invitation_key(*context: aries_cloudagent.config.injection_context.InjectionContext*, *invitation_key: str*, *initiator: str = None*) → aries_cloudagent.messaging.connections.models.connection_record.Co

Retrieve a connection record by invitation key.

Parameters

- **context** – The injection context to use
- **invitation_key** – The key on the originating invitation
- **initiator** – Filter by the initiator value

classmethod retrieve_by_request_id(*context: aries_cloudagent.config.injection_context.InjectionContext*, *request_id: str*) → aries_cloudagent.messaging.connections.models.connection_record.Connect

Retrieve a connection record from our previous request ID.

Parameters

- **context** – The injection context to use
- **request_id** – The ID of the originating connection request

retrieve_invitation(*context: aries_cloudagent.config.injection_context.InjectionContext*) → aries_cloudagent.messaging.connections.messages.connection_invitation.ConnectionInvitation

Retrieve the related connection invitation.

Parameters **context** – The injection context to use

retrieve_request(*context: aries_cloudagent.config.injection_context.InjectionContext*) → aries_cloudagent.messaging.connections.messages.connection_request.ConnectionRequest

Retrieve the related connection invitation.

Parameters **context** – The injection context to use

update_activity_meta(*context: aries_cloudagent.config.injection_context.InjectionContext*, *activity_id: str*, *meta: dict*) → Sequence[dict]

Update metadata for an activity entry.

Parameters

- **context** – The injection context to use
- **activity_id** – The ID of the activity entry
- **meta** – The metadata stored on the activity

updated_activity(*context: aries_cloudagent.config.injection_context.InjectionContext*)

Call webhook when the record activity is updated.

class aries_cloudagent.messaging.connections.models.connection_record.**ConnectionRecordSchem**

Bases: aries_cloudagent.messaging.models.base_record.BaseRecordSchema

Schema to allow serialization/deserialization of connection records.

class Meta

Bases: object

ConnectionRecordSchema metadata.

model_class

alias of *ConnectionRecord*

accept

Used by autodoc_mock_imports.

connection_id

Used by autodoc_mock_imports.

error_msg

Used by autodoc_mock_imports.

inbound_connection_id

Used by autodoc_mock_imports.

initiator

Used by autodoc_mock_imports.

invitation_key

Used by autodoc_mock_imports.

my_did

Used by autodoc_mock_imports.

request_id

Used by autodoc_mock_imports.

routing_state

Used by autodoc_mock_imports.

their_did

Used by autodoc_mock_imports.

their_label

Used by autodoc_mock_imports.

their_role

Used by autodoc_mock_imports.

aries_cloudagent.messaging.connections.models.connection_target module

Record used to handle routing of messages to another agent.

class aries_cloudagent.messaging.connections.models.connection_target.**ConnectionTarget**(*, *did: str = None*, *endpoint: str = None*, *label: str = None*, *recipient_keys: Sequence = None*, *routing_keys: Sequence = None*, *sender_key: str = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Record used to handle routing of messages to another agent.

class Meta

Bases: object

ConnectionTarget metadata.

schema_class = 'ConnectionTargetSchema'

class aries_cloudagent.messaging.connections.models.connection_target.**ConnectionTargetSchema**

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

ConnectionTarget schema.

class Meta

Bases: object

ConnectionTargetSchema metadata.

model_class

alias of *ConnectionTarget*

did
Used by autodoc_mock_imports.

endpoint
Used by autodoc_mock_imports.

label
Used by autodoc_mock_imports.

recipient_keys
Used by autodoc_mock_imports.

routing_keys
Used by autodoc_mock_imports.

sender_key
Used by autodoc_mock_imports.

aries_cloudagent.messaging.connections.tests package

Submodules

aries_cloudagent.messaging.connections.tests.test_connection_record module

class aries_cloudagent.messaging.connections.tests.test_connection_record.**TestConfig**
Bases: object

test_did = '55GkHamhTU1ZbTbV2ab9DE'

test_endpoint = 'http://localhost'

test_seed = 'testseed000000000000000000000001'

test_target_did = 'GbuDUYXaUZRfHD2jeDuQuP'

test_target_verkey = '9WCgWKUaAJj3VWxxtzvvMQN3AoFxoBtBDo9ntwJnVVCC'

test_verkey = '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'

class aries_cloudagent.messaging.connections.tests.test_connection_record.**TestConnectionRe**

Bases: sphinx.ext.autodoc.importer._MockObject, *aries_cloudagent.messaging.connections.tests.test_connection_record.TestConfig*

setUp()

test_active_is_ready()

test_request_is_not_ready()

test_response_is_ready()

test_save_retrieve_compare()

aries_cloudagent.messaging.connections.tests.test_diddoc module

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class aries_cloudagent.messaging.connections.tests.test_diddoc.**TestDIDDoc**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

test_basic()

test_canon_did()

test_embedded_authkey()

test_minimal()

test_minimal_explicit()

test_minimal_ids()

test_missing_recipkey()

test_no_ident()

test_pubkey_type()

test_reference_authkey()

test_w3c_minimal()

Submodules

aries_cloudagent.messaging.connections.manager module

Classes to manage connections.

class aries_cloudagent.messaging.connections.manager.**ConnectionManager**(*context: aries_cloudagent.config.injec*

Bases: object

Class for managing connections.

RECORD_TYPE_DID_DOC = 'did_doc'

RECORD_TYPE_DID_KEY = 'did_key'

accept_response(*response: aries_cloudagent.messaging.connections.messages.connection_response.ConnectionResponse*, *delivery: aries_cloudagent.messaging.message_delivery.MessageDelivery*) → aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord

Accept a connection response.

Process a ConnectionResponse message by looking up the connection request and setting up the pairwise connection.

Parameters

- **response** – The *ConnectionResponse* to accept
- **delivery** – The message delivery metadata

Returns The updated *ConnectionRecord* representing the connection

Raises

- *ConnectionManagerError* – If there is no DID associated with the connection response
- *ConnectionManagerError* – If the corresponding connection is not at the request or response stage

add_key_for_did(*did: str*, *key: str*)
Store a verkey for lookup against a DID.

Parameters

- **did** – The DID to associate with this key
- **key** – The verkey to be added

context
Accessor for the current injection context.

Returns The injection context for this connection manager

create_did_document(*my_info: aries_cloudagent.wallet.base.DIDInfo*, *inbound_connection_id: str = None*, *my_endpoint: str = None*) → aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc
Create our DID document for a given DID.

Parameters

- **my_info** – The DID I am using in this connection
- **inbound_connection_id** – The DID of the inbound routing connection to use
- **my_endpoint** – A custom endpoint for the DID Document

Returns The prepared *DIDDoc* instance

create_invitation(*my_label: str = None*, *my_endpoint: str = None*, *their_role: str = None*, *accept: str = None*, *public: bool = False*) → Tuple[aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord, aries_cloudagent.messaging.connections.messages.connection_invitation.ConnectionInvitation]
Generate new connection invitation.

This interaction represents an out-of-band communication channel. In the future and in practice, these sort of invitations will be received over any number of channels such as SMS, Email, QR Code, NFC, etc.

Structure of an invite message: ‘‘‘json {

“@type”: “did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/invitation”, “label”: “Alice”, “did”: “did:sov:QmWbsNYhMrjHiqZDTUTEJs”

}‘‘‘

Or, in the case of a peer DID: ‘‘‘json {

“@type”: “did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/connections/1.0/invitation”, “label”: “Alice”, “did”: “did:peer:oiSqsNYhMrjHiqZDTUthsw”, “recipientKeys”: [“8HH5gYEeNc3z7PYXmd54d4x6qAfCNrqQqEB3nS7Zfu7K”], “serviceEndpoint”: “https://example.com/endpoint”

}‘‘‘

Currently, only peer DID is supported.

Parameters

- **my_label** – label for this connection
- **my_endpoint** – endpoint where other party can reach me

- **their_role** – a role to assign the connection
- **accept** – set to 'auto' to auto-accept a corresponding connection request
- **public** – set to True to create an invitation from the public DID

Returns A tuple of the new *ConnectionRecord* and *ConnectionInvitation* instances

create_request(*connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord*, *my_label: str = None*, *my_endpoint: str = None*) → aries_cloudagent.messaging.connections.messages.connection_request.ConnectionRequest

Create a new connection request for a previously-received invitation.

Parameters

- **connection** – The *ConnectionRecord* representing the invitation to accept
- **my_label** – My label
- **my_endpoint** – My endpoint

Returns A new *ConnectionRequest* message to send to the other agent

create_response(*connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord*, *my_endpoint: str = None*) → aries_cloudagent.messaging.connections.messages.connection_response.Co

Create a connection response for a received connection request.

Parameters

- **connection** – The *ConnectionRecord* with a pending connection request
- **my_endpoint** – The endpoint I can be reached at

Returns A tuple of the updated *ConnectionRecord* new *ConnectionResponse* message

diddoc_connection_target(*doc: aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc*, *sender_verkey: str*, *their_label: str = None*) → aries_cloudagent.messaging.connections.models.connection_target.ConnectionTarget

Create a connection target from a DID Document.

Parameters

- **doc** – The DID Document to create the target from
- **sender_verkey** – The verkey we are using
- **their_label** – The connection label they are using

establish_inbound(*connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord*, *inbound_connection_id: str*, *outbound_handler*) → str

Assign the inbound routing connection for a connection record.

Returns: the current routing state (request or done)

fetch_did_document(*did: str*) → aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc

Retrieve a DID Document for a given DID.

Parameters **did** – The DID to search for

find_connection(*their_did: str*, *my_did: str = None*, *my_verkey: str = None*, *auto_complete=False*) → aries_cloudagent.messaging.connections.models.connection_record.Connectio

Look up existing connection information for a sender verkey.

Parameters

- **their_did** – Their DID
- **my_did** – My DID

- **my_verkey** – My verkey
- **auto_complete** – Should this connection automatically be promoted to active

Returns The located *ConnectionRecord*, if any

find_did_for_key(*key: str*) → str
Find the DID previously associated with a key.

Parameters **key** – The verkey to look up

find_message_connection(*delivery: aries_cloudagent.messaging.message_delivery.MessageDelivery*) → aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord
Deserialize an incoming message and further populate the request context.

Parameters **delivery** – The message delivery details

Returns The *ConnectionRecord* associated with the expanded message, if any

get_connection_target(*connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRec* → aries_cloudagent.messaging.connections.models.connection_target.ConnectionTarget
Create a connection target from a *ConnectionRecord*.

Parameters **connection** – The connection record (with associated *DIDDoc*) used to generate the connection target

log_activity(*connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord*, *activity_type: str*, *direction: str*, *meta: dict = None*)
Log activity against a connection record and send webhook.

receive_invitation(*invitation: aries_cloudagent.messaging.connections.messages.connection_invitation.ConnectionInvit their_role: str = None*, *accept: str = None*) → aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord
Create a new connection record to track a received invitation.

Parameters

- **invitation** – The *ConnectionInvitation* to store
- **their_role** – The role assigned to this connection
- **accept** – set to 'auto' to auto-accept the invitation

Returns The new *ConnectionRecord* instance

receive_request(*request: aries_cloudagent.messaging.connections.messages.connection_request.ConnectionRequest*, *delivery: aries_cloudagent.messaging.message_delivery.MessageDelivery*) → aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord
Receive and store a connection request.

Parameters

- **request** – The *ConnectionRequest* to accept
- **delivery** – The message delivery metadata

Returns The new or updated *ConnectionRecord* instance

remove_keys_for_did(*did: str*)
Remove all keys associated with a DID.

Parameters **did** – The DID to remove keys for

store_did_document(*did_doc: aries_cloudagent.messaging.connections.models.diddoc.diddoc.DIDDoc*)
Store a DID document.

Parameters **did_doc** – The *DIDDoc* instance to be persisted

update_inbound(*inbound_connection_id: str*, *recip_verkey: str*, *routing_state: str*)
Activate connections once a route has been established.

Looks up pending connections associated with the inbound routing connection and marks the routing as complete.

exception aries_cloudagent.messaging.connections.manager.**ConnectionManagerError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Connection error.

aries_cloudagent.messaging.connections.message_types module

Message type identifiers for Connections.

aries_cloudagent.messaging.connections.routes module

Connection handling admin routes.

class aries_cloudagent.messaging.connections.routes.**ConnectionListSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for connection list.

results
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.connections.routes.**InvitationResultSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for a new connection invitation.

connection_id
Used by autodoc_mock_imports.

invitation
Used by autodoc_mock_imports.

invitation_url
Used by autodoc_mock_imports.

aries_cloudagent.messaging.connections.routes.**connection_sort_key**(*conn*)
Get the sorting key for a particular connection.

aries_cloudagent.messaging.connections.routes.**connections_accept_invitation**(*request: <sphinx.ext.autodoc.i object at 0x7fba4a01d320>*)

Request handler for accepting a stored connection invitation.

Parameters **request** – aiohttp request object

Returns The resulting connection record details

aries_cloudagent.messaging.connections.routes.**connections_accept_request**(*request: <sphinx.ext.autodoc.impo object at 0x7fba4a01d320>*)

Request handler for accepting a stored connection request.

Parameters **request** – aiohttp request object

Returns The resulting connection record details

aries_cloudagent.messaging.connections.routes.**connections_create_invitation**(*request: <sphinx.ext.autodoc.i object at 0x7fba4a01d320>*)

Request handler for creating a new connection invitation.

Parameters **request** – aiohttp request object

Returns The connection invitation details

aries_cloudagent.messaging.connections.routes.**connections_establish_inbound**(*request: <sphinx.ext.autodoc.i object at 0x7fba4a01d320>*)

Request handler for setting the inbound connection on a connection record.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.connections.routes.**connections_list**(*request: <sphinx.ext.autodoc.importer._MockObje object at 0x7fba4a01d320>*)

Request handler for searching connection records.

Parameters **request** – aiohttp request object

Returns The connection list response

aries_cloudagent.messaging.connections.routes.**connections_receive_invitation**(*request: <sphinx.ext.autodoc object at 0x7fba4a01d320>*)

Request handler for receiving a new connection invitation.

Parameters **request** – aiohttp request object

Returns The resulting connection record details

aries_cloudagent.messaging.connections.routes.**connections_remove**(*request: <sphinx.ext.autodoc.importer._MockO object at 0x7fba4a01d320>*)

Request handler for removing a connection record.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.connections.routes.**connections_retrieve**(*request: <sphinx.ext.autodoc.importer._Mo object at 0x7fba4a01d320>*)

Request handler for fetching a single connection record.

Parameters **request** – aiohttp request object

Returns The connection record response

aries_cloudagent.messaging.connections.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a01d320>*)

Register routes.

aries_cloudagent.messaging.credential_definitions package

Submodules

aries_cloudagent.messaging.credential_definitions.routes module

Credential definition admin routes.

class aries_cloudagent.messaging.credential_definitions.routes.**CredentialDefinitionGetResul**

Bases: sphinx.ext.autodoc.importer._MockObject

Results schema for schema get request.

credential_definition
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credential_definitions.routes.**CredentialDefinitionSendRequ**

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for schema send request.

schema_id
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credential_definitions.routes.**CredentialDefinitionSendResu**

Bases: sphinx.ext.autodoc.importer._MockObject

Results schema for schema send request.

credential_definition_id
Used by autodoc_mock_imports.

aries_cloudagent.messaging.credential_definitions.routes.**credential_definitions_get_credent**

Request handler for getting a credential definition from the ledger.

Parameters **request** – aiohttp request object

Returns The credential offer details.

aries_cloudagent.messaging.credential_definitions.routes.**credential_definitions_send_creden**

Request handler for sending a credential definition to the ledger.

Parameters **request** – aiohttp request object

Returns The credential offer details.

aries_cloudagent.messaging.credential_definitions.routes.**register**(*app: <sphinx.ext.autodoc.importer._Mock object at 0x7fba49f8f828>*)

Register routes.

aries_cloudagent.messaging.credentials package

Subpackages

aries_cloudagent.messaging.credentials.handlers package

Submodules

aries_cloudagent.messaging.credentials.handlers.credential_issue_handler module

Basic message handler.

class aries_cloudagent.messaging.credentials.handlers.credential_issue_handler.**CredentialIs**

Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for credential offers.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)

Message handler logic for credential offers.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.credentials.handlers.credential_offer_handler module

Basic message handler.

class aries_cloudagent.messaging.credentials.handlers.credential_offer_handler.**CredentialOf**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for credential offers.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler logic for credential offers.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.credentials.handlers.credential_request_handler module

Credential request handler.

class aries_cloudagent.messaging.credentials.handlers.credential_request_handler.**Credential**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for credential requests.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler logic for credential requests.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.credentials.messages package

Subpackages

aries_cloudagent.messaging.credentials.messages.tests package

Submodules

aries_cloudagent.messaging.credentials.messages.tests.test_credential module

aries_cloudagent.messaging.credentials.messages.tests.test_credential_offer module

aries_cloudagent.messaging.credentials.messages.tests.test_credential_request module

Submodules

aries_cloudagent.messaging.credentials.messages.credential_issue module

A credential content message.

class aries_cloudagent.messaging.credentials.messages.credential_issue.**CredentialIssue**(*, *issue: str = None*, **kw

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a credential.

class Meta

Bases: object

Credential metadata.

handler_class = 'aries_cloudagent.messaging.credentials.handlers.credential_issue_h

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/credential-issuance/0.1/credent

schema_class = 'CredentialIssueSchema'

class aries_cloudagent.messaging.credentials.messages.credential_issue.**CredentialIssueSchem**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Credential schema.

class Meta

Bases: object

Credential schema metadata.

model_class

alias of *CredentialIssue*

issue

Used by autodoc_mock_imports.

aries_cloudagent.messaging.credentials.messages.credential_offer module

A credential offer content message.

class aries_cloudagent.messaging.credentials.messages.credential_offer.**CredentialOffer**(*, *offer_j* *str* = *None* *cre-* *den-* *tial_j* *dict* = *None* *com-* *ment* *str* = *None* ***kw*

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a credential offer.

class Meta

Bases: object

CredentialOffer metadata.

handler_class = 'aries_cloudagent.messaging.credentials.handlers.credential_offer_h

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/credential-issuance/0.1/credent

schema_class = 'CredentialOfferSchema'

class aries_cloudagent.messaging.credentials.messages.credential_offer.**CredentialOfferSchem**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Credential offer schema.

class Meta

Bases: object

Credential offer schema metadata.

model_class

alias of *CredentialOffer*

comment

Used by autodoc_mock_imports.

credential_preview

Used by autodoc_mock_imports.

offer_json

Used by autodoc_mock_imports.

aries_cloudagent.messaging.credentials.messages.credential_request module

A credential request content message.

class aries_cloudagent.messaging.credentials.messages.credential_request.**CredentialRequest**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a credential request.

class Meta

Bases: object

CredentialRequest metadata.

handler_class = 'aries_cloudagent.messaging.credentials.handlers.credential_request

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/credential-issuance/0.1/credent

schema_class = 'CredentialRequestSchema'

class aries_cloudagent.messaging.credentials.messages.credential_request.**CredentialRequestS**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Credential request schema.

class Meta

Bases: object

Credential request schema metadata.

model_class

alias of *CredentialRequest*

comment

Used by autodoc_mock_imports.

request

Used by autodoc_mock_imports.

aries_cloudagent.messaging.credentials.models package

Submodules

aries_cloudagent.messaging.credentials.models.credential_exchange module

Handle credential exchange information interface with non-secrets storage.

class aries_cloudagent.messaging.credentials.models.credential_exchange.**CredentialExchange**

Bases: aries_cloudagent.messaging.models.base_record.BaseRecord

Represents a credential exchange.

INITIATOR_EXTERNAL = 'external'

INITIATOR_SELF = 'self'

LOG_STATE_FLAG = 'debug.credentials'

class Meta

Bases: object

CredentialExchange metadata.

schema_class = 'CredentialExchangeSchema'

RECORD_ID_NAME = 'credential_exchange_id'

RECORD_TYPE = 'credential_exchange'

STATE_CREDENTIAL_RECEIVED = 'credential_received'

STATE_ISSUED = 'issued'

STATE_OFFER_RECEIVED = 'offer_received'

STATE_OFFER_SENT = 'offer_sent'

STATE_REQUEST_RECEIVED = 'request_received'

STATE_REQUEST_SENT = 'request_sent'

STATE_STORED = 'stored'

WEBHOOK_TOPIC = 'credentials'

credential_exchange_id

Accessor for the ID associated with this exchange.

record_tags

Accessor for the record tags generated for this credential exchange.

record_value

Accessor to for the JSON record value props for this credential exchange.

class aries_cloudagent.messaging.credentials.models.credential_exchange.**CredentialExchangeS**

Bases: aries_cloudagent.messaging.models.base_record.BaseRecordSchema

Schema to allow serialization/deserialization of credential exchange records.

class Meta

Bases: object

CredentialExchangeSchema metadata.

model_class

alias of *CredentialExchange*

auto_issue

Used by autodoc_mock_imports.

connection_id

Used by autodoc_mock_imports.

credential

Used by autodoc_mock_imports.

credential_definition_id
Used by autodoc_mock_imports.

credential_exchange_id
Used by autodoc_mock_imports.

credential_id
Used by autodoc_mock_imports.

credential_offer
Used by autodoc_mock_imports.

credential_request
Used by autodoc_mock_imports.

credential_request_metadata
Used by autodoc_mock_imports.

credential_values
Used by autodoc_mock_imports.

error_msg
Used by autodoc_mock_imports.

initiator
Used by autodoc_mock_imports.

parent_thread_id
Used by autodoc_mock_imports.

raw_credential
Used by autodoc_mock_imports.

schema_id
Used by autodoc_mock_imports.

state
Used by autodoc_mock_imports.

thread_id
Used by autodoc_mock_imports.

Submodules

aries_cloudagent.messaging.credentials.manager module

Classes to manage credentials.

class aries_cloudagent.messaging.credentials.manager.**CredentialManager**(*context: aries_cloudagent.config.injec*

Bases: object

Class for managing credentials.

cache_credential_exchange(*credential_exchange_record: aries_cloudagent.messaging.credentials.models.credential_e*
Cache a credential exchange to avoid redundant credential requests.

context
Accessor for the current injection context.

Returns The injection context for this credential manager

create_offer(*credential_definition_id: str*, *connection_id: str*, *auto_issue: bool = None*, *credential_values: dict = None*)
Create a new credential exchange representing an offer.

Parameters

- **credential_definition_id** – Credential definition id for offer
- **connection_id** – Connection to create offer for

Returns A new credential exchange record

create_request(*credential_exchange_record: aries_cloudagent.messaging.credentials.models.credential_exchange.Creden*
connection_record: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecor
Create a credential request.

Parameters

- **credential_exchange_record** – Credential exchange to create request for
- **connection_record** – Connection to create the request for

Returns A tuple (credential_exchange_record, credential_request_message)

credential_stored(*credential_stored_message: aries_cloudagent.messaging.credentials.messages.credential_stored.Cred*
Receive confirmation that holder stored credential.

Parameters **credential_message** – credential to store

issue_credential(*credential_exchange_record: aries_cloudagent.messaging.credentials.models.credential_exchange.Cre*
Issue a credential.

Parameters **credential_exchange_record** – The credential exchange we are issuing a credential for

Returns (Updated credential exchange record, credential message obj)

Return type Tuple

offer_credential(*credential_exchange: aries_cloudagent.messaging.credentials.models.credential_exchange.CredentialE*
Offer a credential.

Parameters **credential_exchange_record** – The credential exchange we are creating the credential offer for

Returns (Updated credential exchange record, credential offer message)

Return type Tuple

perform_send(*credential_exchange: aries_cloudagent.messaging.credentials.models.credential_exchange.CredentialExchan*
outbound_handler)
Send the first message in a credential exchange.

prepare_send(*credential_definition_id: str*, *connection_id: str*, *credential_values: dict*) → aries_cloudagent.messaging.credentials.models.credential_exchange.CredentialExchange
Set up a new credential exchange for an automated send.

Parameters

- **credential_definition_id** – Credential definition id for offer
- **connection_id** – Connection to create offer for
- **credential_values** – The credential values to use if auto_issue is enabled

Returns A new *CredentialExchange* record

receive_credential(*credential_message: aries_cloudagent.messaging.credentials.messages.credential_issue.CredentialIs*
Receive a credential a credential from an issuer.

Hold in storage to be potentially processed by controller before storing.

Parameters **credential_message** – credential to store

receive_offer(*credential_offer_message: aries_cloudagent.messaging.credentials.messages.credential_offer.CredentialOff*
connection_id: str)
Receive a credential offer.

Parameters

- **credential_offer** – Credential offer to receive
- **connection_id** – Connection to receive offer on

Returns The credential_exchange_record

receive_request(*credential_request_message: aries_cloudagent.messaging.credentials.messages.credential_request.Crede*
Receive a credential request.

Parameters **credential_request_message** – Credential request to receive

store_credential(*credential_exchange_record: aries_cloudagent.messaging.credentials.models.credential_exchange.Cred*
Store a credential in the wallet.

Parameters **credential_message** – credential to store

exception aries_cloudagent.messaging.credentials.manager.**CredentialManagerError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Credential error.

aries_cloudagent.messaging.credentials.message_types module

Message type identifiers for Connections.

aries_cloudagent.messaging.credentials.routes module

Connection handling admin routes.

class aries_cloudagent.messaging.credentials.routes.**CredentialExchangeListSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for a credential exchange query.

results
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialIssueRequestSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for sending a credential issue admin message.

credential_values
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialIssueResultSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for sending a credential issue admin message.

credential_id
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialListSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for a credential query.

results
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialOfferRequestSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for sending a credential offer admin message.

connection_id
Used by autodoc_mock_imports.

credential_definition_id
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialOfferResultSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for sending a credential offer admin message.

credential_id
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialProblemReportRequestSchema**(**a
**

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for sending a problem report.

explain_ltxt
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialRequestResultSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for sending a credential request admin message.

credential_id
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for a credential query.

class aries_cloudagent.messaging.credentials.routes.**CredentialSendRequestSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for sending a credential offer admin message.

connection_id
Used by autodoc_mock_imports.

credential_definition_id
Used by autodoc_mock_imports.

credential_values
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.credentials.routes.**CredentialSendResultSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for sending a credential offer admin message.

credential_id
Used by autodoc_mock_imports.

aries_cloudagent.messaging.credentials.routes.**credential_exchange_issue**(*request: <sphinx.ext.autodoc.import ob- ject at 0x7fba49fbdf98>*)

Request handler for sending a credential.

Parameters **request** – aiohttp request object

Returns The credential details.

aries_cloudagent.messaging.credentials.routes.**credential_exchange_list**(*request: <sphinx.ext.autodoc.importe ob- ject at 0x7fba49fbdf98>*)

Request handler for searching credential exchange records.

Parameters **request** – aiohttp request object

Returns The credential exchange list response

aries_cloudagent.messaging.credentials.routes.**credential_exchange_problem_report**(*request: <sphinx.ext.a ob- ject at 0x7fba49fbdf*

Request handler for sending a problem report.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.credentials.routes.**credential_exchange_remove**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for removing a credential exchange record.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.credentials.routes.**credential_exchange_retrieve**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for fetching a single credential exchange record.

Parameters **request** – aiohttp request object

Returns The credential exchange record response

aries_cloudagent.messaging.credentials.routes.**credential_exchange_send**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for sending a credential.

Parameters **request** – aiohttp request object

Returns The credential offer details.

aries_cloudagent.messaging.credentials.routes.**credential_exchange_send_offer**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for sending a credential offer.

Parameters **request** – aiohttp request object

Returns The credential offer details.

aries_cloudagent.messaging.credentials.routes.**credential_exchange_send_request**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for sending a credential request.

Parameters **request** – aiohttp request object

Returns The credential request details.

aries_cloudagent.messaging.credentials.routes.**credential_exchange_store**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for storing a credential request.

Parameters **request** – aiohttp request object

Returns The credential request details.

aries_cloudagent.messaging.credentials.routes.**credentials_get**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for retrieving a credential.

Parameters **request** – aiohttp request object

Returns The credential response

aries_cloudagent.messaging.credentials.routes.**credentials_list**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for searching credential records.

Parameters **request** – aiohttp request object

Returns The credential list response

aries_cloudagent.messaging.credentials.routes.**credentials_remove**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Request handler for searching connection records.

Parameters **request** – aiohttp request object

Returns The connection list response

aries_cloudagent.messaging.credentials.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49fbdf98>*)

Register routes.

aries_cloudagent.messaging.decorators package

Subpackages

aries_cloudagent.messaging.decorators.tests package

Submodules

aries_cloudagent.messaging.decorators.tests.test_decorator_set module

class aries_cloudagent.messaging.decorators.tests.test_decorator_set.**SimpleModel**(*, *value: str = None*, *handled_decorator: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

class Meta

Bases: object

schema_class = 'SimpleModelSchema'

class aries_cloudagent.messaging.decorators.tests.test_decorator_set.**SimpleModelSchema**(**args*, ***kw*

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

class Meta

Bases: object

model_class

alias of *SimpleModel*

handled_decorator = <fields.String(default=<marshmallow.missing>, attribute=None, vali

value = <fields.String(default=<marshmallow.missing>, attribute=None, validate=None, r

class aries_cloudagent.messaging.decorators.tests.test_decorator_set.**TestDecoratorSet**(*metho*

Bases: unittest.case.TestCase

test_decorator_model()

test_dict()

test_extract()

test_field_decorator()

test_skip_decorator()

aries_cloudagent.messaging.decorators.tests.test_thread_decorator module

class aries_cloudagent.messaging.decorators.tests.test_thread_decorator.**TestThreadDecorator**

Bases: unittest.case.TestCase

parent_id = 'tid-000'

received_orders = {'did': 2}

sender_order = 1

test_init()

test_serialize_load()

```
thread_id = 'tid-001'
```

Submodules

aries_cloudagent.messaging.decorators.base module

Classes for managing a collection of decorators.

class aries_cloudagent.messaging.decorators.base.**BaseDecoratorSet**(*models: dict = None*)

Bases: collections.OrderedDict

Collection of decorators.

add_model(*key: str, model: Type[aries_cloudagent.messaging.models.base.BaseModel]*)
Add a registered decorator model.

copy() → aries_cloudagent.messaging.decorators.base.BaseDecoratorSet
Return a copy of the decorator set.

extract_decorators(*message: Mapping[KT, VT_co], schema: Type[<sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a0a9358>] = None, serialized: bool = True, skip_attrs: Sequence[str] = None*) → collections.OrderedDict
Extract decorators and return the remaining properties.

field(*name: str*) → aries_cloudagent.messaging.decorators.base.BaseDecoratorSet
Access a named decorated field.

fields
Acessor for the set of currently defined fields.

has_field(*name: str*) → bool
Check for the existence of a named decorator field.

load_decorator(*key: str*, *value*, *serialized=False*)
Convert a decorator value to its loaded representation.

models
Accessor for the models dictionary.

prefix
Accessor for the decorator prefix.

remove_field(*name: str*)
Remove a named decorated field.

remove_model(*key: str*)
Remove a registered decorator model.

to_dict(*prefix: str = None*) → collections.OrderedDict
Convert to a dictionary (serialize).

Raises BaseModelError – on decorator validation errors

exception aries_cloudagent.messaging.decorators.base.**DecoratorError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Base error for decorator issues.

aries_cloudagent.messaging.decorators.default module

Default decorator set implementation.

class aries_cloudagent.messaging.decorators.default.**DecoratorSet**(*models: dict = None*)

Bases: *aries_cloudagent.messaging.decorators.base.BaseDecoratorSet*

Default decorator set implementation.

aries_cloudagent.messaging.decorators.localization_decorator module

The localization decorator (~l10n) for message localization information.

class aries_cloudagent.messaging.decorators.localization_decorator.**LocalizationDecorator**(**, lo ca st = N lo ca iz- ab Se qu = N ca a- lo Se qu = N*

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing the localization decorator.

class Meta

Bases: object

LocalizationDecorator metadata.

schema_class = 'LocalizationDecoratorSchema'

class aries_cloudagent.messaging.decorators.localization_decorator.**LocalizationDecoratorSc**

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

Localization decorator schema used in serialization/deserialization.

class Meta

Bases: object

LocalizationDecoratorSchema metadata.

model_class

alias of *LocalizationDecorator*

catalogs

Used by autodoc_mock_imports.

locale

Used by autodoc_mock_imports.

localizable

Used by autodoc_mock_imports.

aries_cloudagent.messaging.decorators.signature_decorator module

Model and schema for working with field signatures within message bodies.

class aries_cloudagent.messaging.decorators.signature_decorator.**SignatureDecorator**(*, *signature_type: str = None*, *signature: str = None*, *sig_data: str = None*, *signer: str = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing a field value signed by a known verkey.

class Meta

Bases: object

SignatureDecorator metadata.

schema_class = 'SignatureDecoratorSchema'

TYPE_ED25519SHA512 = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/signature/1.0/ed25519Sha512_

classmethod create(*value*, *signer: str*, *wallet: aries_cloudagent.wallet.base.BaseWallet*, *timestamp=None*) → aries_cloudagent.messaging.decorators.signature_decorator.SignatureDecorator

Create a Signature.

Sign a field value and return a newly constructed *SignatureDecorator* representing the resulting signature.

Parameters

- **value** – Value to sign
- **signer** – Verkey of the signing party
- **wallet** – The wallet to use for the signature

Returns The created *SignatureDecorator* object

decode(*) -> (<class 'object'>, <class 'int'>)

Decode the signature to its timestamp and value.

Returns A tuple of (decoded message, timestamp)

verify(*wallet: aries_cloudagent.wallet.base.BaseWallet*) → bool

Verify the signature against the signer's public key.

Parameters **wallet** – Wallet to use to verify signature

Returns True if verification succeeds else False

class aries_cloudagent.messaging.decorators.signature_decorator.**SignatureDecoratorSchema**(*a, **

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

SignatureDecorator schema.

class Meta

Bases: object

SignatureDecoratorSchema metadata.

model_class

alias of *SignatureDecorator*

sig_data

Used by autodoc_mock_imports.

signature

Used by autodoc_mock_imports.

signature_type

Used by autodoc_mock_imports.

signer

Used by autodoc_mock_imports.

aries_cloudagent.messaging.decorators.thread_decorator module

A message decorator for threads.

A thread decorator identifies a message that may require additional context from previous messages.

class aries_cloudagent.messaging.decorators.thread_decorator.**ThreadDecorator**(*, *thid: str = None*, *pthid: str = None*, *sender_order: int = None*, *received_orders: Mapping[KT, VT_co] = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing thread decorator.

class Meta

Bases: object

ThreadDecorator metadata.

schema_class = 'ThreadDecoratorSchema'

pthid

Accessor for parent thread identifier.

Returns This thread's *pthid*

received_orders

Get received orders.

Returns The highest sender_order value that the sender has seen from other sender(s) on the thread.

sender_order

Get sender order.

Returns A number that tells where this message fits in the sequence of all messages that the current sender has contributed to this thread

thid

Accessor for thread identifier.

Returns This thread's *thid*

class aries_cloudagent.messaging.decorators.thread_decorator.**ThreadDecoratorSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

Thread decorator schema used in serialization/deserialization.

class Meta

Bases: object

ThreadDecoratorSchema metadata.

model_class
alias of *ThreadDecorator*

pthid
Used by autodoc_mock_imports.

received_orders
Used by autodoc_mock_imports.

sender_order
Used by autodoc_mock_imports.

thid
Used by autodoc_mock_imports.

aries_cloudagent.messaging.decorators.timing_decorator module

The timing decorator (~timing).

This decorator allows the timing of agent messages to be communicated and constrained.

class aries_cloudagent.messaging.decorators.timing_decorator.**TimingDecorator**(*, *in_time: Union[str, datetime.datetime] = None*, *out_time: Union[str, datetime.datetime] = None*, *stale_time: Union[str, datetime.datetime] = None*, *expires_time: Union[str, datetime.datetime] = None*, *delay_milli: int = None*, *wait_until_time: Union[str, datetime.datetime] = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing the timing decorator.

class Meta

Bases: object

TimingDecorator metadata.

schema_class = 'TimingDecoratorSchema'

class aries_cloudagent.messaging.decorators.timing_decorator.**TimingDecoratorSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

Timing decorator schema used in serialization/deserialization.

class Meta

Bases: object

TimingDecoratorSchema metadata.

model_class
alias of *TimingDecorator*

delay_milli
Used by autodoc_mock_imports.

expires_time
Used by autodoc_mock_imports.

in_time
Used by autodoc_mock_imports.

out_time
Used by autodoc_mock_imports.

stale_time
Used by autodoc_mock_imports.

wait_until_time
Used by autodoc_mock_imports.

aries_cloudagent.messaging.decorators.transport_decorator module

The transport decorator (~transport).

This decorator allows changes to agent response behaviour and queue status updates.

class aries_cloudagent.messaging.decorators.transport_decorator.**TransportDecorator**(*, *return_route: str = None*, *return_route_ str = None*, *queued_me int = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing the transport decorator.

class Meta
Bases: object

TransportDecorator metadata.

schema_class = 'TransportDecoratorSchema'

class aries_cloudagent.messaging.decorators.transport_decorator.**TransportDecoratorSchema**(*a **

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

Transport decorator schema used in serialization/deserialization.

class Meta
Bases: object

TransportDecoratorSchema metadata.

model_class
alias of *TransportDecorator*

queued_message_count
Used by autodoc_mock_imports.

return_route
Used by autodoc_mock_imports.

return_route_thread
Used by autodoc_mock_imports.

aries_cloudagent.messaging.discovery package

Subpackages

aries_cloudagent.messaging.discovery.handlers package

Subpackages

aries_cloudagent.messaging.discovery.handlers.tests package

Submodules

aries_cloudagent.messaging.discovery.handlers.tests.test_query_handler module

class aries_cloudagent.messaging.discovery.handlers.tests.test_query_handler.**TestQueryHandl**
Bases: object

test_query_all(*request_context*)

aries_cloudagent.messaging.discovery.handlers.tests.test_query_handler.**request_context**() → aries_

Submodules

aries_cloudagent.messaging.discovery.handlers.disclose_handler module

Handler for incoming disclose messages.

class aries_cloudagent.messaging.discovery.handlers.disclose_handler.**DiscloseHandler**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming disclose messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.discovery.handlers.query_handler module

Handler for incoming query messages.

class aries_cloudagent.messaging.discovery.handlers.query_handler.**QueryHandler**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming query messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.discovery.messages package

Subpackages

aries_cloudagent.messaging.discovery.messages.tests package

Submodules

aries_cloudagent.messaging.discovery.messages.tests.test_disclose module

class aries_cloudagent.messaging.discovery.messages.tests.test_disclose.**TestDisclose**(*methodl*
Bases: unittest.case.TestCase

test_deserialize(*mock_disclose_schema_load*)

test_init()

test_protocols = [{'pid': 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/basicmessage/1.0/messa

test_serialize(*mock_disclose_schema_dump*)

test_type()

class aries_cloudagent.messaging.discovery.messages.tests.test_disclose.**TestDiscloseSchema**
Bases: unittest.case.TestCase

disclose = <Disclose(_message_id='3ecce382-867c-4353-8c89-cc5430fc61bd', _message_new_

test_make_model()

aries_cloudagent.messaging.discovery.messages.tests.test_query module

class aries_cloudagent.messaging.discovery.messages.tests.test_query.**TestQuery**(*methodName='r*
Bases: unittest.case.TestCase

test_comment = 'comment'

test_deserialize(*mock_query_schema_load*)

test_init()

test_query = '*'

test_serialize(*mock_query_schema_dump*)

test_type()

class aries_cloudagent.messaging.discovery.messages.tests.test_query.**TestQuerySchema**(*methodl*
Bases: unittest.case.TestCase

query = <Query(_message_id='f79a405e-0812-4fb0-bb74-8abbd535d56d', _message_new_id=Tru

test_make_model()

Submodules

aries_cloudagent.messaging.discovery.messages.disclose module

Represents a protocol discovery disclosure message.

class aries_cloudagent.messaging.discovery.messages.disclose.**Disclose**(*, *protocols: Mapping[str, Mapping[KT, VT_co]] = None*, ***kwargs*)
Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Represents a protocol discovery disclosure, the response to a query message.

class Meta
Bases: object

Disclose metadata.

handler_class = 'aries_cloudagent.messaging.discovery.handlers.disclose_handler.Dis

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/protocol-discovery/1.0/disclose

schema_class = 'DiscloseSchema'

class aries_cloudagent.messaging.discovery.messages.disclose.**DiscloseSchema**(**args*, ***kwargs*)
Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Disclose message schema used in serialization/deserialization.

class Meta
Bases: object

DiscloseSchema metadata.

model_class
alias of *Disclose*

protocols = <fields.List(default=<marshmallow.missing>, attribute=None, validate=None,

class aries_cloudagent.messaging.discovery.messages.disclose.**ProtocolDescriptorSchema**(*only=l ex- clude= many= con- text=N load_c dump_ par- tial=F un- known*

Bases: marshmallow.schema.Schema

Schema for an entry in the protocols list.

opts = <marshmallow.schema.SchemaOpts object>

aries_cloudagent.messaging.discovery.messages.query module

Represents a protocol discovery query message.

class aries_cloudagent.messaging.discovery.messages.query.**Query**(*, *query: str = None*, *comment: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Represents a protocol discovery query.

Used for inspecting what message types are supported by the agent.

class Meta

Bases: object

Query metadata.

handler_class = 'aries_cloudagent.messaging.discovery.handlers.query_handler.QueryH

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/protocol-discovery/1.0/query'

schema_class = 'QuerySchema'

class aries_cloudagent.messaging.discovery.messages.query.**QuerySchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Query message schema used in serialization/deserialization.

class Meta

Bases: object

QuerySchema metadata.

model_class

alias of *Query*

comment = <fields.String(default=<marshmallow.missing>, attribute=None, validate=None,

query = <fields.String(default=<marshmallow.missing>, attribute=None, validate=None, r

Submodules

aries_cloudagent.messaging.discovery.message_types module

Message type identifiers for Protocol Discovery.

aries_cloudagent.messaging.discovery.routes module

Protocol discovery admin routes.

class aries_cloudagent.messaging.discovery.routes.**QueryResultSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for connection list.

results

Used by autodoc_mock_imports.

aries_cloudagent.messaging.discovery.routes.**query_protocols**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f8fef0>*)

Request handler for inspecting supported protocols.

Parameters **request** – aiohttp request object

Returns The diclosed protocols response

aries_cloudagent.messaging.discovery.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f8fef0>*)

Register routes.

aries_cloudagent.messaging.introduction package

Subpackages

aries_cloudagent.messaging.introduction.handlers package

Submodules

aries_cloudagent.messaging.introduction.handlers.forward_invitation_handler module

Handler for incoming forward invitation messages.

class aries_cloudagent.messaging.introduction.handlers.forward_invitation_handler.**ForwardIn**

Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming forward invitation messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)

Message handler implementation.

aries_cloudagent.messaging.introduction.handlers.invitation_handler module

Handler for incoming invitation messages.

class aries_cloudagent.messaging.introduction.handlers.invitation_handler.**InvitationHandler**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming invitation messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.introduction.handlers.invitation_request_handler module

Handler for incoming invitation request messages.

class aries_cloudagent.messaging.introduction.handlers.invitation_request_handler.**Invitatio**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming invitation request messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.introduction.messages package

Subpackages

aries_cloudagent.messaging.introduction.messages.tests package

Submodules

aries_cloudagent.messaging.introduction.messages.tests.test_forward_invitation module

class aries_cloudagent.messaging.introduction.messages.tests.test_forward_invitation.**TestCo**
Bases: object

did = 'did:sov:QmWbsNYhMrjHiqZDTUTEJs'

endpoint_did = 'did:sov:A2wBhNYhMrjHiqZDTUYH7u'

endpoint_url = 'https://example.com/endpoint'

key = '8HH5gYEeNc3z7PYXmd54d4x6qAfCNrqQqEB3nS7Zfu7K'

label = 'Label'

test_message = 'test message'

class aries_cloudagent.messaging.introduction.messages.tests.test_forward_invitation.**TestFo**
Bases: unittest.case.TestCase, *aries_cloudagent.messaging.introduction.messages.tests.test_forward_invitation.TestConfig*

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_invitation_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_serialize(*mock_invitation_schema_dump*)
Test serialization.

test_type()
Test type.

class aries_cloudagent.messaging.introduction.messages.tests.test_forward_invitation.**TestFo**

Bases: sphinx.ext.autodoc.importer._MockObject, *aries_cloudagent.messaging.introduction.messages.tests.test_forward_invitation.TestConfig*

Test forward invitation schema.

test_make_model()

aries_cloudagent.messaging.introduction.messages.tests.test_invitation module

class aries_cloudagent.messaging.introduction.messages.tests.test_invitation.**TestConfig**
Bases: object

did = 'did:sov:QmWbsNYhMrjHiqZDTUTEJs'

endpoint_did = 'did:sov:A2wBhNYhMrjHiqZDTUYH7u'

endpoint_url = 'https://example.com/endpoint'

key = '8HH5gYEeNc3z7PYXmd54d4x6qAfCNrqQqEB3nS7Zfu7K'

label = 'Label'

test_message = 'test message'

class aries_cloudagent.messaging.introduction.messages.tests.test_invitation.**TestInvitation**
Bases: unittest.case.TestCase, *aries_cloudagent.messaging.introduction.messages.tests.test_invitation.TestConfig*

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_invitation_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_serialize(*mock_invitation_schema_dump*)
Test serialization.

test_type()
Test type.

class aries_cloudagent.messaging.introduction.messages.tests.test_invitation.**TestInvitation**

Bases: sphinx.ext.autodoc.importer._MockObject, *aries_cloudagent.messaging.introduction.messages.tests.test_invitation.TestConfig*

Test invitation schema.

test_make_model()

aries_cloudagent.messaging.introduction.messages.tests.test_invitation_request module

class aries_cloudagent.messaging.introduction.messages.tests.test_invitation_request.**TestCo**
Bases: object

test_message = 'MESSAGE'

test_responder = 'RESPONDER'

class aries_cloudagent.messaging.introduction.messages.tests.test_invitation_request.**TestIn**
Bases: unittest.case.TestCase, *aries_cloudagent.messaging.introduction.messages.tests.test_invitation_request.TestConfig*

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_invitation_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_serialize(*mock_invitation_schema_dump*)
Test serialization.

test_type()
Test type.

class aries_cloudagent.messaging.introduction.messages.tests.test_invitation_request.**TestIn**

Bases: sphinx.ext.autodoc.importer._MockObject, *aries_cloudagent.messaging.introduction.messages.tests.test_invitation_request.TestConfig*

Test invitation request schema.

test_make_model()

Submodules

aries_cloudagent.messaging.introduction.messages.forward_invitation module

Represents a forwarded invitation from another agent.

class aries_cloudagent.messaging.introduction.messages.forward_invitation.**ForwardInvitatio**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing an invitation to be forwarded.

class Meta

Bases: object

Metadata for a forwarded invitation.

handler_class = 'aries_cloudagent.messaging.introduction.handlers.forward_invitatio

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/introduction-service/0.1/forwa

schema_class = 'ForwardInvitationSchema'

class aries_cloudagent.messaging.introduction.messages.forward_invitation.**ForwardInvitatio**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

ForwardInvitation request schema class.

class Meta

Bases: object

ForwardInvitation request schema metadata.

model_class

alias of *ForwardInvitation*

invitation

Used by autodoc_mock_imports.

message

Used by autodoc_mock_imports.

aries_cloudagent.messaging.introduction.messages.invitation module

Represents an invitation returned to the introduction service.

class aries_cloudagent.messaging.introduction.messages.invitation.**Invitation**(*, *invitation: aries_cloudagent.m = None*, *message: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing an invitation returned to the introduction service.

class Meta

Bases: object

Metadata for an invitation.

handler_class = 'aries_cloudagent.messaging.introduction.handlers.invitation_handle

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/introduction-service/0.1/invita

schema_class = 'InvitationSchema'

class aries_cloudagent.messaging.introduction.messages.invitation.**InvitationSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Invitation request schema class.

class Meta

Bases: object

Invitation request schema metadata.

model_class

alias of *Invitation*

invitation

Used by autodoc_mock_imports.

message

Used by autodoc_mock_imports.

aries_cloudagent.messaging.introduction.messages.invitation_request module

Represents an request for an invitation from the introduction service.

class aries_cloudagent.messaging.introduction.messages.invitation_request.**InvitationRequest**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing an invitation request.

class Meta

Bases: object

Metadata for an invitation request.

handler_class = 'aries_cloudagent.messaging.introduction.handlers.invitation_reques

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/introduction-service/0.1/invita

schema_class = 'InvitationRequestSchema'

class aries_cloudagent.messaging.introduction.messages.invitation_request.**InvitationRequest**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Invitation request schema class.

class Meta

Bases: object

Invitation request schema metadata.

model_class

alias of *InvitationRequest*

message

Used by autodoc_mock_imports.

responder

Used by autodoc_mock_imports.

aries_cloudagent.messaging.introduction.tests package

Submodules

aries_cloudagent.messaging.introduction.base_service module

Introduction service base classes.

class aries_cloudagent.messaging.introduction.base_service.**BaseIntroductionService**(*context: aries_clou*

Bases: abc.ABC

Service handler for allowing connections to exchange invitations.

return_invitation(*target_connection_id: str*, *invitation: aries_cloudagent.messaging.introduction.messages.invitation.Inv*
outbound_handler)
Handle the forwarding of an invitation to the responder.

Parameters

- **target_connection_id** – The ID of the connection sending the Invitation
- **invitation** – The received Invitation message
- **outbound_handler** – The outbound handler coroutine for sending a message

classmethod service_handler()
Quick accessor for conductor to use.

start_introduction(*init_connection_id: str*, *target_connection_id: str*, *outbound_handler*, *message: str = None*)
Start the introduction process between two connections.

Parameters

- **init_connection_id** – The connection initiating the request
- **target_connection_id** – The connection which is asked for an invitation
- **outbound_handler** – The outbound handler coroutine for sending a message
- **message** – The message to use when requesting the invitation

exception aries_cloudagent.messaging.introduction.base_service.**IntroductionError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Generic introduction service error.

aries_cloudagent.messaging.introduction.demo_service module

Introduction service demo classes.

class aries_cloudagent.messaging.introduction.demo_service.**DemoIntroductionService**(*context: aries_clou*

Bases: *aries_cloudagent.messaging.introduction.base_service.BaseIntroductionService*

Service handler for allowing connections to exchange invitations.

RECORD_TYPE = 'introduction_record'

return_invitation(*target_connection_id: str*, *invitation: aries_cloudagent.messaging.introduction.messages.invitation.Inv*
outbound_handler)
Handle the forwarding of an invitation to the responder.

Parameters

- **target_connection_id** – The ID of the connection sending the Invitation
- **invitation** – The received Invitation message

- **outbound_handler** – The outbound handler coroutine for sending a message

start_introduction(*init_connection_id: str, target_connection_id: str, message: str, outbound_handler*)

Start the introduction process between two connections.

Parameters

- **init_connection_id** – The connection initiating the request
- **target_connection_id** – The connection which is asked for an invitation
- **outbound_handler** – The outbound handler coroutine for sending a message
- **message** – The message to use when requesting the invitation

aries_cloudagent.messaging.introduction.message_types module

Message type identifiers for Introductions.

aries_cloudagent.messaging.introduction.routes module

Introduction service admin routes.

aries_cloudagent.messaging.introduction.routes.**introduction_start**(*request: <sphinx.ext.autodoc.importer._Mock object at 0x7fba4a03db70>*)

Request handler for starting an introduction.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.introduction.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba4a03db70>*)

Register routes.

aries_cloudagent.messaging.models package

Submodules

aries_cloudagent.messaging.models.base module

Base classes for Models and Schemas.

class aries_cloudagent.messaging.models.base.**BaseModel**

Bases: abc.ABC

Base model that provides convenience methods.

class Meta

Bases: object

BaseModel meta data.

schema_class = None

Schema
Accessor for the model's schema class.

Returns The schema class

classmethod deserialize(*obj*)
Convert from JSON representation to a model instance.

Parameters **obj** – The dict to load into a model instance

Returns A model instance for this data

classmethod from_json(*json_repr: Union[str, bytes]*)
Parse a JSON string into a model instance.

Parameters **json_repr** – JSON string

Returns A model instance representation of this JSON

serialize(*as_string=False*) → dict
Create a JSON-compatible dict representation of the model instance.

Parameters **as_string** – Return a string of JSON instead of a dict

Returns A dict representation of this model, or a JSON string if as_string is True

to_json() → str
Create a JSON representation of the model instance.

Returns A JSON representation of this message

exception aries_cloudagent.messaging.models.base.**BaseModelError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.error.BaseError*

Base exception class for base model errors.

class aries_cloudagent.messaging.models.base.**BaseModelSchema**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

BaseModel schema.

class Meta
Bases: object

BaseModelSchema metadata.

model_class = None

ordered = True

skip_values = [None]

Model
Accessor for the schema's model class.

Returns The model class

make_model(*data: dict*)
Return model instance after loading.

Returns A model instance

remove_skipped_values(*data*)
Remove values that are are marked to skip.

Returns Returns this modified data

skip_dump_only(*data*)
Skip fields that are only expected during serialization.

Parameters **data** – The incoming data to clean

Returns The modified data

aries_cloudagent.messaging.models.base.**resolve_class**(*the_cls*, *relative_cls: type = None*)
Resolve a class.

Parameters

- **the_cls** – The class to resolve
- **relative_cls** – Relative class to resolve from

Returns The resolved class

Raises ClassNotFoundError – If the class could not be loaded

aries_cloudagent.messaging.models.base.**resolve_meta_property**(*obj*, *prop_name: str*, *defval=None*)
Resolve a meta property.

Parameters

- **prop_name** – The property to resolve
- **defval** – The default value

Returns The meta property

aries_cloudagent.messaging.presentations package

Subpackages

aries_cloudagent.messaging.presentations.handlers package

Submodules

aries_cloudagent.messaging.presentations.handlers.credential_presentation_handler module

Basic message handler.

class aries_cloudagent.messaging.presentations.handlers.credential_presentation_handler.**Cre**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for credential presentations.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler logic for credential presentations.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.presentations.handlers.presentation_request_handler module

Basic message handler.

class aries_cloudagent.messaging.presentations.handlers.presentation_request_handler.**Prese**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Message handler class for presentation requests.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler logic for presentation requests.

Parameters

- **context** – request context
- **responder** – responder callback

aries_cloudagent.messaging.presentations.messages package

Subpackages

aries_cloudagent.messaging.presentations.messages.tests package

Submodules

aries_cloudagent.messaging.presentations.messages.tests.test_proof module

aries_cloudagent.messaging.presentations.messages.tests.test_proof_request module

Submodules

aries_cloudagent.messaging.presentations.messages.credential_presentation module

A credential presentation message.

class aries_cloudagent.messaging.presentations.messages.credential_presentation.**CredentialP**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a credential presentation.

class Meta
Bases: object

CredentialPresentation metadata.

handler_class = 'aries_cloudagent.messaging.presentations.handlers.credential_prese

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/credential-presentation/0.1/cre

schema_class = 'CredentialPresentationSchema'

class aries_cloudagent.messaging.presentations.messages.credential_presentation.**CredentialP**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

CredentialPresentation schema.

class Meta

Bases: object

CredentialPresentationSchema metadata.

model_class

alias of *CredentialPresentation*

comment

Used by autodoc_mock_imports.

presentation

Used by autodoc_mock_imports.

aries_cloudagent.messaging.presentations.messages.presentation_request module

A presentation request content message.

class aries_cloudagent.messaging.presentations.messages.presentation_request.**PresentationRe**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a presentation request.

class Meta

Bases: object

PresentationRequest metadata.

handler_class = 'aries_cloudagent.messaging.presentations.handlers.presentation_req

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/credential-presentation/0.1/pre

schema_class = 'PresentationRequestSchema'

class aries_cloudagent.messaging.presentations.messages.presentation_request.**PresentationRe**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

PresentationRequest schema.

class Meta

Bases: object

PresentationRequestSchema metadata.

model_class
alias of *PresentationRequest*

comment
Used by autodoc_mock_imports.

request
Used by autodoc_mock_imports.

aries_cloudagent.messaging.presentations.models package

Submodules

aries_cloudagent.messaging.presentations.models.presentation_exchange module

Handle presentation exchange information interface with non-secrets storage.

class aries_cloudagent.messaging.presentations.models.presentation_exchange.**PresentationExc**

Bases: aries_cloudagent.messaging.models.base_record.BaseRecord

Represents a presentation exchange.

INITIATOR_EXTERNAL = 'external'

INITIATOR_SELF = 'self'

LOG_STATE_FLAG = 'debug.presentations'

class Meta

Bases: object

PresentationExchange metadata.

schema_class = 'PresentationExchangeSchema'

RECORD_ID_NAME = 'presentation_exchange_id'

RECORD_TYPE = 'presentation_exchange'

STATE_PRESENTATION_RECEIVED = 'presentation_received'

STATE_PRESENTATION_SENT = 'presentation_sent'

STATE_REQUEST_RECEIVED = 'request_received'

STATE_REQUEST_SENT = 'request_sent'

STATE_VERIFIED = 'verified'

WEBHOOK_TOPIC = 'presentations'

presentation_exchange_id

Accessor for the ID associated with this exchange.

record_tags

Accessor for the record tags generated for this presentation exchange.

record_value

Accessor for JSON record value generated for this presentation exchange.

class aries_cloudagent.messaging.presentations.models.presentation_exchange.**PresentationExc**

Bases: aries_cloudagent.messaging.models.base_record.BaseRecordSchema

Schema for serialization/deserialization of presentation exchange records.

class Meta

Bases: object

PresentationExchangeSchema metadata.

model_class

alias of *PresentationExchange*

connection_id

Used by autodoc_mock_imports.

error_msg

Used by autodoc_mock_imports.

initiator

Used by autodoc_mock_imports.

presentation

Used by autodoc_mock_imports.

presentation_exchange_id
Used by autodoc_mock_imports.

presentation_request
Used by autodoc_mock_imports.

state
Used by autodoc_mock_imports.

thread_id
Used by autodoc_mock_imports.

verified
Used by autodoc_mock_imports.

Submodules

aries_cloudagent.messaging.presentations.manager module

Classes to manage presentations.

class aries_cloudagent.messaging.presentations.manager.**PresentationManager**(*context: aries_cloudagent.confi*

Bases: object

Class for managing presentations.

context
Accessor for the current request context.

Returns The injection context for this presentation manager

create_presentation(*presentation_exchange_record: aries_cloudagent.messaging.presentations.models.presentation_exc requested_credentials: dict*)
Receive a presentation request.

Parameters

- **presentation_exchange_record** – Record to update
- **requested_credentials** – Indy formatted requested_credentials

i.e.,

{

“self_attested_attributes”: { “j233ffbc-bd35-49b1-934f-51e083106f6d”: “value”

}, “requested_attributes”: {

“6253ffbb-bd35-49b3-934f-46e083106f6c”: { “cred_id”: “5bfa40b7-062b-4ae0-a251-a86c87922c0e”, “revealed”: true

}

}, “requested_predicates”: {

“bfc8a97d-60d3-4f21-b998-85eeabe5c8c0”: { “cred_id”: “5bfa40b7-062b-4ae0-a251-a86c87922c0e”

}

}

}

create_request(*name: str*, *version: str*, *requested_attributes: list*, *requested_predicates: list*, *connection_id: str*)
Create a proof request.

receive_presentation(*presentation: dict*, *thread_id: str*)
Receive a presentation request.

receive_request(*presentation_request_message: aries_cloudagent.messaging.presentations.messages.presentation_reques*
connection_id: str)
Receive a presentation request.

Parameters **presentation_request_message** – Presentation message to receive

verify_presentation(*presentation_exchange_record: aries_cloudagent.messaging.presentations.models.presentation_ex*
Verify a presentation request.

exception aries_cloudagent.messaging.presentations.manager.**PresentationManagerError**(**args*, *error_code: str = None*, ***kwargs*

Bases: *aries_cloudagent.error.BaseError*

Presentation error.

aries_cloudagent.messaging.presentations.message_types module

Message type identifiers for Connections.

aries_cloudagent.messaging.presentations.routes module

Admin routes for presentations.

class aries_cloudagent.messaging.presentations.routes.**PresentationExchangeListSchema**(**args*, ***kwarg*

Bases: sphinx.ext.autodoc.importer._MockObject

Result schema for a presentation exchange query.

results
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.presentations.routes.**PresentationRequestRequestSchema**(**args*, ***kw*

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for sending a proof request.

class RequestedAttribute(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

RequestedAttribute model.

name
Used by autodoc_mock_imports.

restrictions
Used by autodoc_mock_imports.

class RequestedPredicate(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

RequestedPredicate model.

name
Used by autodoc_mock_imports.

p_type
Used by autodoc_mock_imports.

p_value
Used by autodoc_mock_imports.

restrictions
Used by autodoc_mock_imports.

connection_id
Used by autodoc_mock_imports.

name
Used by autodoc_mock_imports.

requested_attributes
Used by autodoc_mock_imports.

requested_predicates
Used by autodoc_mock_imports.

version
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.presentations.routes.**SendPresentationRequestSchema**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for sending a presentation.

requested_attributes
Used by autodoc_mock_imports.

requested_predicates
Used by autodoc_mock_imports.

self_attested_attributes
Used by autodoc_mock_imports.

aries_cloudagent.messaging.presentations.routes.**presentation_exchange_create_request**(*request: <sphinx object at 0x7fba4*

Request handler for creating a presentation request.

Parameters **request** – aiohttp request object

Returns The presentation exchange details.

aries_cloudagent.messaging.presentations.routes.**presentation_exchange_credentials_list**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fb*

Request handler for searching applicable credential records.

Parameters **request** – aiohttp request object

Returns The credential list response

aries_cloudagent.messaging.presentations.routes.**presentation_exchange_list**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f7aa58>*)

Request handler for searching presentation exchange records.

Parameters **request** – aiohttp request object

Returns The presentation exchange list response

aries_cloudagent.messaging.presentations.routes.**presentation_exchange_remove**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f7aa58>*)

Request handler for removing a presentation exchange record.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.presentations.routes.**presentation_exchange_retrieve**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f7aa58>*)

Request handler for fetching a single presentation exchange record.

Parameters **request** – aiohttp request object

Returns The presentation exchange record response

aries_cloudagent.messaging.presentations.routes.**presentation_exchange_send_credential_prese**

Request handler for sending a credential presentation.

Parameters **request** – aiohttp request object

Returns The presentation exchange details.

aries_cloudagent.messaging.presentations.routes.**presentation_exchange_send_request**(*request: <sphinx.ex ob- ject at 0x7fba49f7*

Request handler for creating and sending a presentation request.

Parameters **request** – aiohttp request object

Returns The presentation exchange details.

aries_cloudagent.messaging.presentations.routes.**presentation_exchange_verify_credential_pre**

Request handler for verifying a presentation request.

Parameters **request** – aiohttp request object

Returns The presentation exchange details.

aries_cloudagent.messaging.presentations.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f7aa58>*)

Register routes.

aries_cloudagent.messaging.problem_report package

Submodules

aries_cloudagent.messaging.problem_report.handler module

Generic problem report handler.

class aries_cloudagent.messaging.problem_report.handler.**ProblemReportHandler**

Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Problem report handler class.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)

Handle problem report message.

Parameters

- **context** – Request context
- **responder** – Responder used to reply

aries_cloudagent.messaging.problem_report.message module

Represents a generic problem report message.

class aries_cloudagent.messaging.problem_report.message.**ProblemReport**(*, *msg_catalog: str = None*, *locale: str = None*, *explain_ltxt: str = None*, *explain_l10n: Mapping[str, str] = None*, *problem_items: Sequence[Mapping[str, str]] = None*, *who_retries: str = None*, *fix_hint_ltxt: Mapping[str, str] = None*, *impact: str = None*, *where: str = None*, *time_noticed: str = None*, *tracking_uri: str = None*, *escalation_uri: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Base class representing a generic problem report message.

class Meta
Bases: object

Problem report metadata.

handler_class = 'aries_cloudagent.messaging.problem_report.handler.ProblemReportHar

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/notification/1.0/problem-report

schema_class = 'ProblemReportSchema'

class aries_cloudagent.messaging.problem_report.message.**ProblemReportSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Schema for ProblemReport base class.

class Meta
Bases: object

Problem report schema metadata.

model_class
alias of *ProblemReport*

escalation_uri
Used by autodoc_mock_imports.

explain_l10n
Used by autodoc_mock_imports.

explain_ltxt
Used by autodoc_mock_imports.

fix_hint_ltxt
Used by autodoc_mock_imports.

impact
Used by autodoc_mock_imports.

locale
Used by autodoc_mock_imports.

msg_catalog
Used by autodoc_mock_imports.

problem_items
Used by autodoc_mock_imports.

time_noticed
Used by autodoc_mock_imports.

tracking_uri
Used by autodoc_mock_imports.

where
Used by autodoc_mock_imports.

who_retries
Used by autodoc_mock_imports.

aries_cloudagent.messaging.routing package

Subpackages

aries_cloudagent.messaging.routing.handlers package

Subpackages

aries_cloudagent.messaging.routing.handlers.tests package

Submodules

aries_cloudagent.messaging.routing.handlers.tests.test_query_update_handlers module

class aries_cloudagent.messaging.routing.handlers.tests.test_query_update_handlers.**TestQue**
Bases: object

test_no_connection(*request_context*)

test_query_none(*request_context*)

test_query_route(*request_context*)

aries_cloudagent.messaging.routing.handlers.tests.test_query_update_handlers.**request_conte**

Submodules

aries_cloudagent.messaging.routing.handlers.forward_handler module

Handler for incoming forward messages.

class aries_cloudagent.messaging.routing.handlers.forward_handler.**ForwardHandler**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming forward messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.routing.handlers.route_query_request_handler module

Handler for incoming route-query-request messages.

class aries_cloudagent.messaging.routing.handlers.route_query_request_handler.**RouteQueryRe**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming route-query-request messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.routing.handlers.route_query_response_handler module

Handler for incoming route-query-response messages.

class aries_cloudagent.messaging.routing.handlers.route_query_response_handler.**RouteQueryRe**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming route-query-response messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.routing.handlers.route_update_request_handler module

Handler for incoming route-update-request messages.

class aries_cloudagent.messaging.routing.handlers.route_update_request_handler.**RouteUpdateR**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming route-update-request messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.routing.handlers.route_update_response_handler module

Handler for incoming route-update-response messages.

class aries_cloudagent.messaging.routing.handlers.route_update_response_handler.**RouteUpdate**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Handler for incoming route-update-response messages.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Message handler implementation.

aries_cloudagent.messaging.routing.messages package

Subpackages

aries_cloudagent.messaging.routing.messages.tests package

Submodules

aries_cloudagent.messaging.routing.messages.tests.test_forward module

class aries_cloudagent.messaging.routing.messages.tests.test_forward.**TestForward**(*methodName=*
Bases: unittest.case.TestCase

msg = 'msg'

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*message_schema_load*)

test_init()

test_serialize(*message_schema_dump*)

test_type()

to = 'to'

class aries_cloudagent.messaging.routing.messages.tests.test_forward.**TestForwardSchema**(*meth*
Bases: unittest.case.TestCase

test_make_model()

aries_cloudagent.messaging.routing.messages.tests.test_route_query_request module

class aries_cloudagent.messaging.routing.messages.tests.test_route_query_request.**TestRoute**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*message_schema_load*)

test_filter = {'recipient_key': '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'}

test_init()

test_limit = 100

test_offset = 10

test_serialize(*message_schema_dump*)

test_type()

test_verkey = '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'

class aries_cloudagent.messaging.routing.messages.tests.test_route_query_request.**TestRoute**
Bases: unittest.case.TestCase

test_make_model()

aries_cloudagent.messaging.routing.messages.tests.test_route_query_response module

class aries_cloudagent.messaging.routing.messages.tests.test_route_query_response.**TestRoute**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_conn_id = 'conn-id'

test_deserialize(*message_schema_load*)

test_end = 15

test_init()

test_limit = 5

test_route_id = 'route-id'

test_serialize(*message_schema_dump*)

test_start = 10

test_total = 20

test_type()

test_verkey = '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'

class aries_cloudagent.messaging.routing.messages.tests.test_route_query_response.**TestRoute**
Bases: unittest.case.TestCase

test_make_model()

aries_cloudagent.messaging.routing.messages.tests.test_route_update_request module

class aries_cloudagent.messaging.routing.messages.tests.test_route_update_request.**TestRoute**
Bases: unittest.case.TestCase

test_make_model()

class aries_cloudagent.messaging.routing.messages.tests.test_route_update_request.**TestRoute**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_action = 'create'

test_deserialize(*message_schema_load*)

test_init()

test_serialize(*message_schema_dump*)

test_type()

test_verkey = '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'

aries_cloudagent.messaging.routing.messages.tests.test_route_update_response module

class aries_cloudagent.messaging.routing.messages.tests.test_route_update_response.**TestRout**
Bases: unittest.case.TestCase

test_make_model()

class aries_cloudagent.messaging.routing.messages.tests.test_route_update_response.**TestRout**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_action = 'create'

test_deserialize(*message_schema_load*)

test_init()

test_result = 'success'

test_serialize(*message_schema_dump*)

test_type()

test_verkey = '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'

Submodules

aries_cloudagent.messaging.routing.messages.forward module

Represents a forward message.

class aries_cloudagent.messaging.routing.messages.forward.**Forward**(*, *to: str = None*, *msg: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Represents a request to forward a message to a connected agent.

class Meta

Bases: object

Forward metadata.

handler_class = 'aries_cloudagent.messaging.routing.handlers.forward_handler.Forwar

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/routing/1.0/forward'

schema_class = 'ForwardSchema'

class aries_cloudagent.messaging.routing.messages.forward.**ForwardSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Forward message schema used in serialization/deserialization.

class Meta

Bases: object

ForwardSchema metadata.

model_class

alias of *Forward*

msg

Used by autodoc_mock_imports.

to

Used by autodoc_mock_imports.

aries_cloudagent.messaging.routing.messages.route_query_request module

Query existing forwarding routes.

class aries_cloudagent.messaging.routing.messages.route_query_request.**RouteQueryRequest**(*, *filter: dict* = *None*, *paginate* = *None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Query existing routes from a routing agent.

class Meta

Bases: object

RouteQueryRequest metadata.

handler_class = 'aries_cloudagent.messaging.routing.handlers.route_query_request_ha

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/routing/1.0/route-query-request

schema_class = 'RouteQueryRequestSchema'

class aries_cloudagent.messaging.routing.messages.route_query_request.**RouteQueryRequestSche**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

RouteQueryRequest message schema used in serialization/deserialization.

class Meta

Bases: object

RouteQueryRequestSchema metadata.

model_class

alias of *RouteQueryRequest*

filter = <fields.Dict(default=<marshmallow.missing>, attribute=None, validate=None, re

paginate = <fields.Nested(default=<marshmallow.missing>, attribute=None, validate=None

aries_cloudagent.messaging.routing.messages.route_query_response module

Return existing forwarding routes in response to a query.

class aries_cloudagent.messaging.routing.messages.route_query_response.**RouteQueryResponse**(*

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Return existing routes from a routing agent.

class Meta

Bases: object

RouteQueryResponse metadata.

handler_class = 'aries_cloudagent.messaging.routing.handlers.route_query_response_h

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/routing/1.0/route-query-respons

schema_class = 'RouteQueryResponseSchema'

class aries_cloudagent.messaging.routing.messages.route_query_response.**RouteQueryResponseSc**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

RouteQueryResponse message schema used in serialization/deserialization.

class Meta

Bases: object

RouteQueryResponseSchema metadata.

model_class

alias of *RouteQueryResponse*

paginated = <fields.Nested(default=<marshmallow.missing>, attribute=None, validate=Non

routes = <fields.List(default=<marshmallow.missing>, attribute=None, validate=None, re

aries_cloudagent.messaging.routing.messages.route_update_request module

Request to update forwarding routes.

class aries_cloudagent.messaging.routing.messages.route_update_request.**RouteUpdateRequest**(*

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Request to existing routes with a routing agent.

class Meta
Bases: object

RouteUpdateRequest metadata.

handler_class = 'aries_cloudagent.messaging.routing.handlers.route_update_request_h

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/routing/1.0/route-update-reques

schema_class = 'RouteUpdateRequestSchema'

class aries_cloudagent.messaging.routing.messages.route_update_request.**RouteUpdateRequestSc**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

RouteUpdateRequest message schema used in serialization/deserialization.

class Meta
Bases: object

RouteUpdateRequestSchema metadata.

model_class
alias of *RouteUpdateRequest*

updates
Used by autodoc_mock_imports.

aries_cloudagent.messaging.routing.messages.route_update_response module

Response for a route update request.

class aries_cloudagent.messaging.routing.messages.route_update_response.**RouteUpdateResponse**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Response for a route update request.

class Meta
Bases: object

RouteUpdateResponse metadata.

handler_class = 'aries_cloudagent.messaging.routing.handlers.route_update_response_

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/routing/1.0/route-update-respon

schema_class = 'RouteUpdateResponseSchema'

class aries_cloudagent.messaging.routing.messages.route_update_response.**RouteUpdateResponse**

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

RouteUpdateResponse message schema used in serialization/deserialization.

class Meta
Bases: object

RouteUpdateResponseSchema metadata.

model_class
alias of *RouteUpdateResponse*

updated = <fields.List(default=<marshmallow.missing>, attribute=None, validate=None, r

aries_cloudagent.messaging.routing.models package

Submodules

aries_cloudagent.messaging.routing.models.paginate module

An object for containing the request pagination information.

class aries_cloudagent.messaging.routing.models.paginate.**Paginate**(*, *limit: int = None*, *offset: int = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing the pagination details of a request.

class Meta
Bases: object

Paginate metadata.

schema_class = 'PaginateSchema'

class aries_cloudagent.messaging.routing.models.paginate.**PaginateSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

Paginate schema.

class Meta
Bases: object

PaginateSchema metadata.

model_class = 'Paginate'

limit = <fields.Integer(default=<marshmallow.missing>, attribute=None, validate=None,

offset = <fields.Integer(default=<marshmallow.missing>, attribute=None, validate=None,

aries_cloudagent.messaging.routing.models.paginated module

An object for containing the response pagination information.

class aries_cloudagent.messaging.routing.models.paginated.**Paginated**(*, *start: int = None*, *end: int = None*, *limit: int = None*, *total: int = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing the pagination details of a response.

class Meta

Bases: object

Paginated metadata.

schema_class = 'PaginatedSchema'

class aries_cloudagent.messaging.routing.models.paginated.**PaginatedSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

Paginated schema.

class Meta

Bases: object

PaginatedSchema metadata.

model_class = 'Paginated'

end = <fields.Integer(default=<marshmallow.missing>, attribute=None, validate=None, re

limit = <fields.Integer(default=<marshmallow.missing>, attribute=None, validate=None,

start = <fields.Integer(default=<marshmallow.missing>, attribute=None, validate=None,

total = <fields.Integer(default=<marshmallow.missing>, attribute=None, validate=None,

aries_cloudagent.messaging.routing.models.route_query_result module

An object for containing returned route information.

class aries_cloudagent.messaging.routing.models.route_query_result.**RouteQueryResult**(*, *recipient_key: str = None*, ***kwargs*

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing route information returned by a route query.

class Meta

Bases: object

RouteQueryResult metadata.

schema_class = 'RouteQueryResultSchema'

class aries_cloudagent.messaging.routing.models.route_query_result.**RouteQueryResultSchema**(*

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

RouteQueryResult schema.

class Meta

Bases: object

RouteQueryResultSchema metadata.

model_class = 'RouteQueryResult'

recipient_key = <fields.String(default=<marshmallow.missing>, attribute=None, validate=

aries_cloudagent.messaging.routing.models.route_record module

An object for containing information on an individual route.

class aries_cloudagent.messaging.routing.models.route_record.**RouteRecord**(*, *record_id: str = None*, *connection_id: str = None*, *recipient_key: str = None*, *created_at: str = None*, *updated_at: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing stored route information.

class Meta

Bases: object

RouteRecord metadata.

schema_class = 'RouteRecordSchema'

class aries_cloudagent.messaging.routing.models.route_record.**RouteRecordSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

RouteRecord schema.

class Meta

Bases: object

RouteRecordSchema metadata.

model_class = 'RouteRecord'

connection_id

Used by autodoc_mock_imports.

created_at

Used by autodoc_mock_imports.

recipient_key

Used by autodoc_mock_imports.

record_id

Used by autodoc_mock_imports.

updated_at

Used by autodoc_mock_imports.

aries_cloudagent.messaging.routing.models.route_update module

An object for containing route information to be updated.

class aries_cloudagent.messaging.routing.models.route_update.**RouteUpdate**(**, recipient_key: str = None, action: str = None, **kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing a route update request.

ACTION_CREATE = 'create'

ACTION_DELETE = 'delete'

class Meta

Bases: object

RouteUpdate metadata.

schema_class = 'RouteUpdateSchema'

class aries_cloudagent.messaging.routing.models.route_update.**RouteUpdateSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

RouteUpdate schema.

class Meta

Bases: object

RouteUpdateSchema metadata.

model_class = 'RouteUpdate'

action

Used by autodoc_mock_imports.

recipient_key

Used by autodoc_mock_imports.

aries_cloudagent.messaging.routing.models.route_updated module

An object for containing updated route information.

class aries_cloudagent.messaging.routing.models.route_updated.**RouteUpdated**(**, recipient_key: str = None, action: str = None, result: str = None, **kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Class representing a route update response.

class Meta

Bases: object

RouteUpdated metadata.

schema_class = 'RouteUpdatedSchema'

RESULT_CLIENT_ERROR = 'client_error'

RESULT_NO_CHANGE = 'no_change'

RESULT_SERVER_ERROR = 'server_error'

RESULT_SUCCESS = 'success'

class aries_cloudagent.messaging.routing.models.route_updated.**RouteUpdatedSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

RouteUpdated schema.

class Meta

Bases: object

RouteUpdatedSchema metadata.

model_class = 'RouteUpdated'

action

Used by autodoc_mock_imports.

recipient_key

Used by autodoc_mock_imports.

result

Used by autodoc_mock_imports.

aries_cloudagent.messaging.routing.tests package

Submodules

aries_cloudagent.messaging.routing.tests.test_routing_manager module

class aries_cloudagent.messaging.routing.tests.test_routing_manager.**TestRoutingManager**

Bases: object

test_create_delete(*manager*)

test_create_retrieve(*manager*)

test_no_recipient(*manager*)

test_retrieve_none(*manager*)

aries_cloudagent.messaging.routing.tests.test_routing_manager.**manager**() → aries_cloudagent.messaging.ro

aries_cloudagent.messaging.routing.tests.test_routing_manager.**request_context**() → aries_cloudagent.m

Submodules

aries_cloudagent.messaging.routing.manager module

Routing manager classes for tracking and inspecting routing records.

exception aries_cloudagent.messaging.routing.manager.**RouteNotFoundError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.routing.manager.RoutingManagerError*

Requested route was not found.

class aries_cloudagent.messaging.routing.manager.**RoutingManager**(*context: aries_cloudagent.config.injection_conte*

Bases: object

Class for handling routing records.

RECORD_TYPE = 'forward_route'

context
Accessor for the current request context.

Returns The request context for this connection

create_route_record(*client_connection_id: str = None*, *recipient_key: str = None*) → aries_cloudagent.messaging.routing.models.route_record.RouteRecord
Create and store a new RouteRecord.

Parameters

- **client_connection_id** – The ID of the connection record
- **recipient_key** – The recipient verkey of the route

Returns The new routing record

delete_route_record(*route: aries_cloudagent.messaging.routing.models.route_record.RouteRecord*)
Remove an existing route record.

get_recipient(*recip_verkey: str*) → aries_cloudagent.messaging.routing.models.route_record.RouteRecord
Resolve the recipient for a verkey.

Parameters **recip_verkey** – The verkey ("to") of the incoming Forward message

Returns The *RouteRecord* associated with this verkey

get_routes(*client_connection_id: str = None*, *tag_filter: dict = None*) → Sequence[aries_cloudagent.messaging.routing.models.route_record.RouteRecord]
Fetch all routes associated with the current connection.

Parameters

- **client_connection_id** – The ID of the connection record
- **tag_filter** – An optional dictionary of tag filters

Returns A sequence of route records found by the query

send_create_route(*router_connection_id: str*, *recip_key: str*, *outbound_handler*)
Create and send a route update request.

Returns: the current routing state (request or done)

update_routes(*client_connection_id: str, updates: Sequence[aries_cloudagent.messaging.routing.models.route_update.Rou* → Sequence[aries_cloudagent.messaging.routing.models.route_updated.RouteUpdated]
Update routes associated with the current connection.

Parameters

- **client_connection_id** – The ID of the connection record
- **updates** – The sequence of route updates (create/delete) to perform.

exception aries_cloudagent.messaging.routing.manager.**RoutingManagerError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Generic routing error.

aries_cloudagent.messaging.routing.message_types module

Message type identifiers for Routing.

aries_cloudagent.messaging.schemas package

Submodules

aries_cloudagent.messaging.schemas.routes module

Credential schema admin routes.

class aries_cloudagent.messaging.schemas.routes.**SchemaGetResultsSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Results schema for schema get request.

schema_json
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.schemas.routes.**SchemaSendRequestSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Request schema for schema send request.

attributes
Used by autodoc_mock_imports.

schema_name
Used by autodoc_mock_imports.

schema_version
Used by autodoc_mock_imports.

class aries_cloudagent.messaging.schemas.routes.**SchemaSendResultsSchema**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Results schema for schema send request.

schema_id
Used by autodoc_mock_imports.

aries_cloudagent.messaging.schemas.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f8f470>*)

Register routes.

aries_cloudagent.messaging.schemas.routes.**schemas_get_schema**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f8f470>*)

Request handler for sending a credential offer.

Parameters **request** – aiohttp request object

Returns The credential offer details.

aries_cloudagent.messaging.schemas.routes.**schemas_send_schema**(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f8f470>*)

Request handler for sending a credential offer.

Parameters **request** – aiohttp request object

Returns The credential offer details.

aries_cloudagent.messaging.tests package

Submodules

aries_cloudagent.messaging.tests.test_agent_message module

class aries_cloudagent.messaging.tests.test_agent_message.**BasicAgentMessage**(*_id: str = None, _decorators: aries_cloudagent.mes = None*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Simple agent message implementation

class Meta
Bases: object

Meta data

message_type = 'basic-message'

schema_class = 'AgentMessageSchema'

class aries_cloudagent.messaging.tests.test_agent_message.**SignedAgentMessage**(*value: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Signed agent message tests

class Meta

Bases: object

Meta data

handler_class = None

message_type = 'signed-agent-message'

schema_class = 'SignedAgentMessageSchema'

class aries_cloudagent.messaging.tests.test_agent_message.**SignedAgentMessageSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Utility schema

class Meta

Bases: object

model_class

alias of *SignedAgentMessage*

signed_fields = ('value',)

value = <fields.String(default=<marshmallow.missing>, attribute=None, validate=None, r

class aries_cloudagent.messaging.tests.test_agent_message.**TestAgentMessage**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Tests agent message.

class BadImplementationClass(*_id: str = None*, *_decorators: aries_cloudagent.messaging.decorators.base.BaseDecoratorSet = None*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Test utility class.

test_assign_thread()

test_field_signature()

test_init()

Tests init class

aries_cloudagent.messaging.tests.test_protocol_registry module

class aries_cloudagent.messaging.tests.test_protocol_registry.**TestProtocolRegistry**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

no_type_message = {'a': 'b'}

setUp()

test_disclosed()

test_message_handler = 'fake_handler'

test_message_type = 'PROTOCOL/MESSAGE'

test_message_type_query()

test_protocol = 'PROTOCOL'

test_protocol_queries = ['*', 'PROTOCOL', 'PROTO*']

test_protocol_queries_fail = ['', 'nomatch', 'nomatch*']

test_protocols()

unknown_type_message = {'@type': 1}

aries_cloudagent.messaging.tests.test_utils module

class aries_cloudagent.messaging.tests.test_utils.**TestUtils**(*methodName='runTest'*)

Bases: unittest.case.TestCase

test_format()

test_parse()

aries_cloudagent.messaging.trustping package

Subpackages

aries_cloudagent.messaging.trustping.handlers package

Submodules

aries_cloudagent.messaging.trustping.handlers.ping_handler module

Ping handler.

class aries_cloudagent.messaging.trustping.handlers.ping_handler.**PingHandler**

Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Ping handler class.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)

Handle ping message.

Parameters

- **context** – Request context
- **responder** – Responder used to reply

aries_cloudagent.messaging.trustping.handlers.ping_response_handler module

Ping response handler.

class aries_cloudagent.messaging.trustping.handlers.ping_response_handler.**PingResponseHandl**
Bases: *aries_cloudagent.messaging.base_handler.BaseHandler*

Ping response handler class.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Handle ping response message.

Parameters

- **context** – Request context
- **responder** – Responder used to reply

aries_cloudagent.messaging.trustping.messages package

Subpackages

aries_cloudagent.messaging.trustping.messages.tests package

Submodules

aries_cloudagent.messaging.trustping.messages.tests.test_trust_ping module

class aries_cloudagent.messaging.trustping.messages.tests.test_trust_ping.**TestPing**(*methodNa*
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_ping_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_serialize(*mock_ping_schema_load*)
Test serialization.

test_type()
Test type.

class aries_cloudagent.messaging.trustping.messages.tests.test_trust_ping.**TestPingSchema**(*
**

Bases: sphinx.ext.autodoc.importer._MockObject

Test ping schema.

test_make_model()

aries_cloudagent.messaging.trustping.messages.tests.test_trust_ping_reponse module

class aries_cloudagent.messaging.trustping.messages.tests.test_trust_ping_reponse.**TestPingR**
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_deserialize(*mock_ping_schema_load*)
Test deserialization.

test_init()
Test initialization.

test_serialize(*mock_ping_schema_load*)
Test serialization.

test_type()
Test type.

class aries_cloudagent.messaging.trustping.messages.tests.test_trust_ping_reponse.**TestPingS**

Bases: sphinx.ext.autodoc.importer._MockObject

Test ping response schema.

test_make_model()

Submodules

aries_cloudagent.messaging.trustping.messages.ping module

Represents a trust ping message.

class aries_cloudagent.messaging.trustping.messages.ping.**Ping**(*, *response_requested: bool = True*, *comment: str = None*, ***kwargs*)
Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a trustping message.

class Meta
Bases: object

Ping metadata.

handler_class = 'aries_cloudagent.messaging.trustping.handlers.ping_handler.PingHan

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/trust_ping/1.0/ping'

schema_class = 'PingSchema'

class aries_cloudagent.messaging.trustping.messages.ping.**PingSchema**(**args*, ***kwargs*)
Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

Schema for Ping class.

class Meta
Bases: object

PingSchema metadata.

model_class
alias of *Ping*

comment
Used by autodoc_mock_imports.

response_requested
Used by autodoc_mock_imports.

aries_cloudagent.messaging.trustping.messages.ping_response module

Represents an response to a trust ping message.

class aries_cloudagent.messaging.trustping.messages.ping_response.**PingResponse**(*, *comment: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.agent_message.AgentMessage*

Class representing a ping response.

class Meta
Bases: object

PingResponse metadata.

handler_class = 'aries_cloudagent.messaging.trustping.handlers.ping_response_handle

message_type = 'did:sov:BzCbsNYhMrjHiqZDTUASHg;spec/trust_ping/1.0/ping_response'

schema_class = 'PingResponseSchema'

class aries_cloudagent.messaging.trustping.messages.ping_response.**PingResponseSchema**(**args*, ***kwarg*

Bases: *aries_cloudagent.messaging.agent_message.AgentMessageSchema*

PingResponse schema.

class Meta
Bases: object

PingResponseSchema metadata.

model_class
alias of *PingResponse*

comment = <fields.String(default=<marshmallow.missing>, attribute=None, validate=None,

Submodules

aries_cloudagent.messaging.trustping.message_types module

Message type identifiers for Trust Pings.

aries_cloudagent.messaging.trustping.routes module

Trust ping admin routes.

aries_cloudagent.messaging.trustping.routes.**connections_send_ping**(*request: <sphinx.ext.autodoc.importer._Mock object at 0x7fba49f9ad68>*)

Request handler for sending a trust ping to a connection.

Parameters **request** – aiohttp request object

aries_cloudagent.messaging.trustping.routes.**register**(*app: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba49f9ad68>*)

Register routes.

Submodules

aries_cloudagent.messaging.agent_message module

Agent message base class and schema.

class aries_cloudagent.messaging.agent_message.**AgentMessage**(*_id: str = None*, *_decorators: aries_cloudagent.messaging.decorators.base. = None*)

Bases: *aries_cloudagent.messaging.models.base.BaseModel*

Agent message base class.

Handler

Accessor for the agent message's handler class.

Returns Handler class

class Meta

Bases: object

AgentMessage metadata.

handler_class = None

message_type = None

schema_class = None

assign_thread_from(*msg: aries_cloudagent.messaging.agent_message.AgentMessage*)

Copy thread information from a previous message.

Parameters **msg** – The received message containing optional thread information

assign_thread_id(*thid: str*, *pthid: str = None*)

Assign a specific thread ID.

Parameters

- **thid** – The thread identifier
- **pthid** – The parent thread identifier

get_signature(*field_name: str*) → aries_cloudagent.messaging.decorators.signature_decorator.SignatureDecorator

Get the signature for a named field.

Parameters **field_name** – Field name to get the signature for

Returns A SignatureDecorator for the requested field name

set_signature(*field_name: str*, *signature: aries_cloudagent.messaging.decorators.signature_decorator.SignatureDecorator*
Add or replace the signature for a named field.

Parameters

- **field_name** – Field to set signature on
- **signature** – Signature for the field

sign_field(*field_name: str*, *signer_verkey: str*, *wallet: aries_cloudagent.wallet.base.BaseWallet*, *timestamp=None*) → aries_cloudagent.messaging.decorators.signature_decorator.SignatureDecorator
Create and store a signature for a named field.

Parameters

- **field_name** – Field to sign
- **signer_verkey** – Verkey of signer
- **wallet** – Wallet to use for signature
- **timestamp** – Optional timestamp for signature

Returns A SignatureDecorator for newly created signature

Raises ValueError – If field_name doesn't exist on this message

verify_signatures(*wallet: aries_cloudagent.wallet.base.BaseWallet*) → bool
Verify all associated field signatures.

Parameters **wallet** – Wallet to use in verification

Returns True if all signatures verify, else false

verify_signed_field(*field_name: str*, *wallet: aries_cloudagent.wallet.base.BaseWallet*, *signer_verkey: str = None*) → str
Verify a specific field signature.

Parameters

- **field_name** – The field name to verify
- **wallet** – Wallet to use for the verification
- **signer_verkey** – Verkey of signer to use

Returns The verkey of the signer

Raises

- ValueError – If field_name does not exist on this message
- ValueError – If the verification fails
- ValueError – If the verkey of the signature does not match the
- provided verkey

exception aries_cloudagent.messaging.agent_message.**AgentMessageError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelError*

Base exception for agent message issues.

class aries_cloudagent.messaging.agent_message.**AgentMessageSchema**(**args*, ***kwargs*)

Bases: *aries_cloudagent.messaging.models.base.BaseModelSchema*

AgentMessage schema.

class Meta

Bases: object

AgentMessageSchema metadata.

model_class = None

signed_fields = None

check_dump_decorators(*obj*)

Pre-dump hook to validate and load the message decorators.

Parameters **obj** – The AgentMessage object

Raises BaseModelError – If a decorator does not validate

dump_decorators(*data*)

Post-dump hook to write the decorators to the serialized output.

Parameters **obj** – The serialized data

Returns The modified data

extract_decorators(*data*)

Pre-load hook to extract the decorators and check the signed fields.

Parameters **data** – Incoming data to parse

Returns Parsed and modified data

Raises

- ValidationError – If a field signature does not correlate
- to a field in the message
- ValidationError – If the message defines both a field signature
- and a value for the same field
- ValidationError – If there is a missing field signature

populate_decorators(*obj*)

Post-load hook to populate decorators on the message.

Parameters **obj** – The AgentMessage object

Returns The AgentMessage object with populated decorators

replace_signatures(*data*)

Post-dump hook to write the signatures to the serialized output.

Parameters **obj** – The serialized data

Returns The modified data

aries_cloudagent.messaging.base_context module

Abstract RequestContext base class.

class aries_cloudagent.messaging.base_context.**BaseRequestContext**
Bases: abc.ABC

Abstract RequestContext base class.

This class is only for resolving recursive import issues.

aries_cloudagent.messaging.base_handler module

A Base handler class for all message handlers.

class aries_cloudagent.messaging.base_handler.**BaseHandler**
Bases: abc.ABC

Abstract base class for handlers.

handle(*context: aries_cloudagent.messaging.request_context.RequestContext*, *responder: aries_cloudagent.messaging.responder.BaseResponder*)
Abstract method for handler logic.

Parameters

- **context** – Request context object
- **responder** – A responder object

exception aries_cloudagent.messaging.base_handler.**HandlerException**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.error.BaseError*

Exception base class for generic handler errors.

aries_cloudagent.messaging.error module

Messaging-related error classes and codes.

exception aries_cloudagent.messaging.error.**MessageParseError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.error.BaseError*

Message parse error.

error_code = 'message_parse_error'

exception aries_cloudagent.messaging.error.**MessagePrepareError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.error.BaseError*

Message preparation error.

error_code = 'message_prepare_error'

aries_cloudagent.messaging.message_delivery module

Classes for representing message delivery details.

class aries_cloudagent.messaging.message_delivery.**MessageDelivery**(*, *connection_id: str = None*, *direct_response: bool = False*, *direct_response_requested: str = None*, *in_time: datetime.datetime = None*, *raw_message: str = None*, *recipient_verkey: str = None*, *recipient_did: str = None*, *recipient_did_public: str = None*, *sender_did: str = None*, *sender_verkey: str = None*, *socket_id: str = None*, *thread_id: str = None*, *transport_type: str = None*)

Bases: object

Properties of an agent message's delivery.

connection_id
Accessor for the pairwise connection identifier.

Returns This context's connection identifier

direct_response
Accessor for the flag indicating that direct responses are preferred.

Returns This context's direct response flag

direct_response_requested
Accessor for the requested direct response mode.

Returns This context's requested direct response mode

in_time
Accessor for the datetime the message was received.

Returns This context's received time

raw_message
Accessor for the raw message text.

Returns The raw message text

recipient_did
Accessor for the recipient DID which corresponds with the verkey.

Returns The recipient DID

recipient_did_public
Check if the recipient did is public.

Indicates whether the message is associated with a public (ledger) recipient DID.

Returns True if the recipient's DID is public, else false

recipient_verkey
Accessor for the recipient verkey key used to pack the incoming request.

Returns The recipient verkey

sender_did
Accessor for the sender DID which corresponds with the verkey.

Returns The sender did

sender_verkey
Accessor for the sender public key used to pack the incoming request.

Returns This context's sender's verkey

socket_id
Accessor for the identifier of the incoming socket connection.

Returns This context's socket identifier

thread_id
Accessor for the identifier of the message thread.

Returns The delivery thread ID

transport_type
Accessor for the transport type used to receive the message.

Returns This context's transport type

aries_cloudagent.messaging.outbound_message module

Outbound message representation.

class aries_cloudagent.messaging.outbound_message.**OutboundMessage**(*payload: Union[str, bytes], *, connection_id: str = None, encoded: bool = False, endpoint: str = None, reply_socket_id: str = None, reply_thread_id: str = None, reply_to_verkey: str = None, target: aries_cloudagent.messaging.connect = None*)

Bases: object

Represents an outgoing message.

endpoint
Return the endpoint of the outbound message.

Defaults to the endpoint of the connection target.

aries_cloudagent.messaging.protocol_registry module

Handle registration and publication of supported message families.

class aries_cloudagent.messaging.protocol_registry.**ProtocolRegistry**
Bases: object

Protocol registry for indexing message families.

controllers
Accessor for a list of all protocol controller functions.

message_types
Accessor for a list of all message types.

prepare_disclosed(*context: aries_cloudagent.config.injection_context.InjectionContext, protocols: Sequence[str]*)
Call controllers and return publicly supported message families and roles.

protocols
Accessor for a list of all message protocols.

protocols_matching_query(*query: str*) → Sequence[str]
Return a list of message protocols matching a query string.

register_controllers(**controller_sets*)
Add new controllers.

Parameters **controller_sets** – Mappings of message families to coroutines

register_message_types(**typesets*)
Add new supported message types.

Parameters **typesets** – Mappings of message types to register

resolve_message_class(*message_type: str*) → type
Resolve a message_type to a message class.

Given a message type identifier, this method returns the corresponding registered message class.

Parameters **message_type** – Message type to resolve

Returns The resolved message class

aries_cloudagent.messaging.request_context module

Request context class.

A request context provides everything required by handlers and other parts of the system to process a message.

class aries_cloudagent.messaging.request_context.**RequestContext**(*, *base_context: aries_cloudagent.config.injection_conte = None*, *settings: Mapping[str, object] = None*)
Bases: *aries_cloudagent.config.injection_context.InjectionContext*

Context established by the Conductor and passed into message handlers.

connection_ready
Accessor for the flag indicating an active connection with the sender.

Returns True if the connection is active, else False

connection_record
Accessor for the related connection record.

copy() → aries_cloudagent.messaging.request_context.RequestContext
Produce a copy of the request context instance.

default_endpoint
Accessor for the default agent endpoint (from agent config).

Returns The default agent endpoint

default_label
Accessor for the default agent label (from agent config).

Returns The default label

message
Accessor for the deserialized message instance.

Returns This context's agent message

message_delivery
Accessor for the message delivery information.

Returns This context's message delivery information

aries_cloudagent.messaging.responder module

A message responder.

The responder is provided to message handlers to enable them to send a new message in response to the message being handled.

class aries_cloudagent.messaging.responder.**BaseResponder**(*, *connection_id: str = None*, *reply_socket_id: str = None*, *reply_to_verkey: str = None*)

Bases: abc.ABC

Interface for message handlers to send responses.

create_outbound(*message: Union[aries_cloudagent.messaging.agent_message.AgentMessage, str, bytes]*, *, *connection_id: str = None*, *reply_socket_id: str = None*, *reply_thread_id: str = None*, *reply_to_verkey: str = None*, *target: aries_cloudagent.messaging.connections.models.connection_target.ConnectionTarget = None*) → aries_cloudagent.messaging.outbound_message.OutboundMessage

Create an OutboundMessage from a message payload.

send(*message: Union[aries_cloudagent.messaging.agent_message.AgentMessage, str, bytes]*, ***kwargs*)

Convert a message to an OutboundMessage and send it.

send_outbound(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)

Send an outbound message.

Parameters **message** – The *OutboundMessage* to be sent

send_reply(*message: Union[aries_cloudagent.messaging.agent_message.AgentMessage, str, bytes]*, *, *connection_id: str = None*, *target: aries_cloudagent.messaging.connections.models.connection_target.ConnectionTarget = None*)

Send a reply to an incoming message.

Parameters

- **message** – the *AgentMessage*, or pre-packed str or bytes to reply with
- **connection_id** – optionally override the target connection ID
- **target** – optionally specify a *ConnectionTarget* to send to

Raises *ResponderError* – If there is no active connection

send_webhook(*topic: str*, *payload: dict*)

Dispatch a webhook.

Parameters

- **topic** – the webhook topic identifier
- **payload** – the webhook payload value

class aries_cloudagent.messaging.responder.**MockResponder**

Bases: *aries_cloudagent.messaging.responder.BaseResponder*

Mock responder implementation for use by tests.

send(*message: Union[aries_cloudagent.messaging.agent_message.AgentMessage, str, bytes]*, ***kwargs*)

Convert a message to an OutboundMessage and send it.

send_outbound(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)
Send an outbound message.

send_reply(*message: Union[aries_cloudagent.messaging.agent_message.AgentMessage, str, bytes], **kwargs*)
Send a reply to an incoming message.

send_webhook(*topic: str*, *payload: dict*)
Send an outbound message.

exception aries_cloudagent.messaging.responder.**ResponderError**(**args*, *error_code: str = None*, ***kwargs*)
Bases: *aries_cloudagent.error.BaseError*

Responder error.

aries_cloudagent.messaging.serializer module

Standard message serializer classes.

class aries_cloudagent.messaging.serializer.**MessageSerializer**
Bases: object

Standard DIDComm message parser and serializer.

encode_message(*context: aries_cloudagent.config.injection_context.InjectionContext, message_json: Union[str, bytes], recipient_keys: Sequence[str], routing_keys: Sequence[str], sender_key: str*) → Union[str, bytes]
Encode an outgoing message for transport.

Parameters

- **context** – The injection context for settings and services
- **message_json** – The message body to serialize
- **recipient_keys** – A sequence of recipient verkeys
- **routing_keys** – A sequence of routing verkeys
- **sender_key** – The verification key of the sending agent

Returns The encoded message

extract_message_type(*parsed_msg: dict*) → str
Extract the message type identifier from a parsed message.

Raises MessageParseError – If the message doesn't specify a type

parse_message(*context: aries_cloudagent.config.injection_context.InjectionContext, message_body: Union[str, bytes], transport_type: str*) → Tuple[dict, aries_cloudagent.messaging.message_delivery.MessageDelivery]
Deserialize an incoming message and further populate the request context.

Parameters

- **context** – The injection context for settings and services
- **message_body** – The body of the message
- **transport_type** – The transport the message was received on

Returns A message delivery object with details on the parsed message

Raises

- `MessageParseError` – If the JSON parsing failed
- `MessageParseError` – If a wallet is required but can't be located

aries_cloudagent.messaging.socket module

Duplex connection handling classes.

class aries_cloudagent.messaging.socket.**SocketInfo**(*, *connection_id: str = None*, *handler: Coroutine[T_co, T_contra, V_co] = None*, *reply_mode: str = None*, *reply_thread_ids: Sequence[str] = None*, *reply_verkeys: Sequence[str] = None*, *single_response: _asyncio.Future = None*, *socket_id: str = None*)

Bases: `object`

Track an open transport connection for direct routing of outbound messages.

REPLY_MODE_ALL = 'all'

REPLY_MODE_NONE = 'none'

REPLY_MODE_THREAD = 'thread'

add_reply_thread_id(*thid: str*)
Add a thread ID to the set of potential reply targets.

add_reply_verkey(*verkey: str*)
Add a verkey to the set of potential reply targets.

closed
Accessor for the socket closed state.

dispatch_complete()
Indicate that a message handler has completed.

process_incoming(*parsed_msg: dict*, *delivery: aries_cloudagent.messaging.message_delivery.MessageDelivery*)
Process an incoming message and update the socket metadata as necessary.

Parameters

- **parsed_msg** – The unserialized message body
- **delivery** – The message delivery metadata

reply_mode
Accessor for the socket reply mode.

select_outgoing(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*) → bool
Determine if an outbound message should be sent to this socket.

Parameters **message** – The outbound message to be checked

send(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)
.

class aries_cloudagent.messaging.socket.**SocketRef**(*socket_id: str, close: Coroutine[T_co, T_contra, V_co]*)

Bases: object

A reference to a registered duplex connection.

aries_cloudagent.messaging.util module

Utils for messages.

aries_cloudagent.messaging.util.**datetime_now**() → datetime.datetime
Timestamp in UTC.

aries_cloudagent.messaging.util.**datetime_to_str**(*dt: Union[str, datetime.datetime]*) → str
Convert a datetime object to an indy-standard datetime string.

Parameters **dt** – May be a string or datetime to allow automatic conversion

aries_cloudagent.messaging.util.**str_to_datetime**(*dt: Union[str, datetime.datetime]*) → datetime.datetime
Convert an indy-standard datetime string to a datetime.

Using a fairly lax regex pattern to match slightly different formats. In Python 3.7 datetime.fromisoformat might be used.

Parameters **dt** – May be a string or datetime to allow automatic conversion

aries_cloudagent.messaging.util.**time_now**() → str
Timestamp in ISO format.

1.1.9 aries_cloudagent.storage package

Subpackages

aries_cloudagent.storage.tests package

Storage test suite

Submodules

aries_cloudagent.storage.tests.test_basic_storage module

class aries_cloudagent.storage.tests.test_basic_storage.**TestBasicStorage**
Bases: object

test_add_id_required(*store*)

test_add_required(*store*)

test_add_retrieve(*store*)

test_closed_search(*store*)

test_delete(*store*)

test_delete_missing(*store*)

test_delete_tags(*store*)

test_delete_tags_missing(*store*)

test_iter_search(*store*)

test_retrieve_missing(*store*)

test_search(*store*)

test_update_missing(*store*)

test_update_tags(*store*)

test_update_tags_missing(*store*)

test_update_value(*store*)

aries_cloudagent.storage.tests.test_basic_storage.**store**()

aries_cloudagent.storage.tests.test_basic_storage.**test_missing_record**(*tags={}*)

aries_cloudagent.storage.tests.test_basic_storage.**test_record**(*tags={}*)

aries_cloudagent.storage.tests.test_indy_storage module

aries_cloudagent.storage.tests.test_indy_storage.**store**()

aries_cloudagent.storage.tests.test_storage_record module

class aries_cloudagent.storage.tests.test_storage_record.**TestStorageRecord**

Bases: object

test_create()

Submodules

aries_cloudagent.storage.base module

Abstract base classes for non-secrets storage.

class aries_cloudagent.storage.base.**BaseStorage**

Bases: abc.ABC

Abstract Non-Secrets interface.

add_record(*record: aries_cloudagent.storage.record.StorageRecord*)

Add a new record to the store.

Parameters **record** – *StorageRecord* to be stored

delete_record(*record: aries_cloudagent.storage.record.StorageRecord*)

Delete an existing record.

Parameters **record** – *StorageRecord* to delete

delete_record_tags(*record: aries_cloudagent.storage.record.StorageRecord*, *tags: (typing.Sequence, typing.Mapping)*)

Update an existing stored record's tags.

Parameters

- **record** – *StorageRecord* to delete

- **tags** – Tags

get_record(*record_type: str*, *record_id: str*) → aries_cloudagent.storage.record.StorageRecord
Fetch a record from the store by type and ID.

Parameters

- **record_type** – The record type
- **record_id** – The record id

Returns A *StorageRecord* instance

search_records(*type_filter: str*, *tag_query: Mapping[KT, VT_co] = None*, *page_size: int = None*) → aries_cloudagent.storage.base.BaseStorageRecordSearch
Create a new record query.

Parameters

- **type_filter** – Filter string
- **tag_query** – Tags to query
- **page_size** – Page size

Returns An instance of *BaseStorageRecordSearch*

update_record_tags(*record: aries_cloudagent.storage.record.StorageRecord*, *tags: Mapping[KT, VT_co]*)
Update an existing stored record's tags.

Parameters

- **record** – *StorageRecord* to update
- **tags** – New tags

update_record_value(*record: aries_cloudagent.storage.record.StorageRecord*, *value: str*)
Update an existing stored record's value.

Parameters

- **record** – *StorageRecord* to update
- **value** – The new value

class aries_cloudagent.storage.base.**BaseStorageRecordSearch**(*store: aries_cloudagent.storage.base.BaseStorage*, *type_filter: str*, *tag_query: Mapping[KT, VT_co]*, *page_size: int = None*)
Bases: abc.ABC

Represent an active stored records search.

close()
Dispose of the search query.

fetch(*max_count: int*) → Sequence[aries_cloudagent.storage.record.StorageRecord]
Fetch the next list of results from the store.

Parameters **max_count** – Max number of records to return

Returns A list of 'StorageRecord's

fetch_all() → Sequence[aries_cloudagent.storage.record.StorageRecord]
Fetch all records from the query.

fetch_single() → aries_cloudagent.storage.record.StorageRecord
Fetch a single query result.

handle
Handle a search request.

open()
Start the search query.

opened
Accessor for open state.

Returns True if opened, else False

page_size
Accessor for page size.

Returns The page size

store
BaseStorage backend for this implementation.

Returns The *BaseStorage* implementation being used

tag_query
Accessor for tag query.

Returns The tag query

type_filter
Accessor for type filter.

Returns The type filter

aries_cloudagent.storage.basic module

Basic in-memory storage implementation (non-wallet).

class aries_cloudagent.storage.basic.**BasicStorage**(*_wallet: aries_cloudagent.wallet.base.BaseWallet = None*)
Bases: *aries_cloudagent.storage.base.BaseStorage*

Basic in-memory storage class.

add_record(*record: aries_cloudagent.storage.record.StorageRecord*)
Add a new record to the store.

Parameters **record** – *StorageRecord* to be stored

Raises

- StorageError – If no record is provided
- StorageError – If the record has no ID

delete_record(*record: aries_cloudagent.storage.record.StorageRecord*)
Delete a record.

Parameters **record** – *StorageRecord* to delete

Raises StorageNotFoundError – If record not found

delete_record_tags(*record: aries_cloudagent.storage.record.StorageRecord*, *tags: (typing.Sequence, typing.Mapping)*)
Update an existing stored record's tags.

Parameters

- **record** – *StorageRecord* to delete
- **tags** – Tags

Raises `StorageNotFoundError` – If record not found

get_record(*record_type: str*, *record_id: str*) → aries_cloudagent.storage.record.StorageRecord
Fetch a record from the store by type and ID.

Parameters

- **record_type** – The record type
- **record_id** – The record id

Returns A *StorageRecord* instance

Raises `StorageNotFoundError` – If the record is not found

search_records(*type_filter: str*, *tag_query: Mapping[KT, VT_co] = None*, *page_size: int = None*) → aries_cloudagent.storage.basic.BasicStorageRecordSearch
Search stored records.

Parameters

- **type_filter** – Filter string
- **tag_query** – Tags to query
- **page_size** – Page size

Returns An instance of *BaseStorageRecordSearch*

update_record_tags(*record: aries_cloudagent.storage.record.StorageRecord, tags: Mapping[KT, VT_co]*)
Update an existing stored record's tags.

Parameters

- **record** – *StorageRecord* to update
- **tags** – New tags

Raises `StorageNotFoundError` – If record not found

update_record_value(*record: aries_cloudagent.storage.record.StorageRecord*, *value: str*)
Update an existing stored record's value.

Parameters

- **record** – *StorageRecord* to update
- **value** – The new value

Raises `StorageNotFoundError` – If record not found

class aries_cloudagent.storage.basic.**BasicStorageRecordSearch**(*store: aries_cloudagent.storage.basic.BasicStora type_filter: str, tag_query: Mapping[KT, VT_co], page_size: int = None*)

Bases: *aries_cloudagent.storage.base.BaseStorageRecordSearch*

Represent an active stored records search.

close()

Dispose of the search query.

fetch(*max_count: int*) → Sequence[aries_cloudagent.storage.record.StorageRecord]

Fetch the next list of results from the store.

Parameters **max_count** – Max number of records to return

Returns A list of 'StorageRecord's

Raises StorageSearchError – If the search query has not been opened

open()

Start the search query.

opened

Accessor for open state.

Returns True if opened, else False

aries_cloudagent.storage.basic.**basic_tag_query_match**(*tags: dict, tag_query: dict*) → bool

Match simple tag filters (string values).

aries_cloudagent.storage.basic.**basic_tag_value_match**(*value: str, match: dict*) → bool

Match a single tag against a tag subquery.

TODO: What type coercion is needed? (support int or float values?)

aries_cloudagent.storage.error module

Storage-related exceptions.

exception aries_cloudagent.storage.error.**StorageDuplicateError**(**args, error_code: str = None, **kwargs*)

Bases: *aries_cloudagent.storage.error.StorageError*

Duplicate record found in storage.

exception aries_cloudagent.storage.error.**StorageError**(**args, error_code: str = None, **kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Base class for Storage errors.

exception aries_cloudagent.storage.error.**StorageNotFoundError**(**args, error_code: str = None, **kwargs*)

Bases: *aries_cloudagent.storage.error.StorageError*

Record not found in storage.

exception aries_cloudagent.storage.error.**StorageSearchError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.storage.error.StorageError*

General exception during record search.

aries_cloudagent.storage.indy module

Indy implementation of BaseStorage interface.

class aries_cloudagent.storage.indy.**IndyStorage**(*wallet: aries_cloudagent.wallet.indy.IndyWallet*)

Bases: *aries_cloudagent.storage.base.BaseStorage*

Indy Non-Secrets interface.

add_record(*record: aries_cloudagent.storage.record.StorageRecord*)
Add a new record to the store.

Parameters **record** – *StorageRecord* to be stored

delete_record(*record: aries_cloudagent.storage.record.StorageRecord*)
Delete a record.

Parameters **record** – *StorageRecord* to delete

Raises

- StorageNotFoundError – If record not found
- StorageError – If a libindy error occurs

delete_record_tags(*record: aries_cloudagent.storage.record.StorageRecord*, *tags: (typing.Sequence, typing.Mapping)*)
Update an existing stored record's tags.

Parameters

- **record** – *StorageRecord* to delete
- **tags** – Tags

get_record(*record_type: str*, *record_id: str*) → aries_cloudagent.storage.record.StorageRecord
Fetch a record from the store by type and ID.

Parameters

- **record_type** – The record type
- **record_id** – The record id

Returns A *StorageRecord* instance

Raises

- StorageError – If the record is not provided
- StorageError – If the record ID not provided
- StorageNotFoundError – If the record is not found
- StorageError – If record not found

search_records(*type_filter: str*, *tag_query: Mapping[KT, VT_co] = None*, *page_size: int = None*) → aries_cloudagent.storage.indy.IndyStorageRecordSearch

Search stored records.

Parameters

- **type_filter** – Filter string
- **tag_query** – Tags to query
- **page_size** – Page size

Returns An instance of *BaseStorageRecordSearch*

update_record_tags(*record: aries_cloudagent.storage.record.StorageRecord*, *tags: Mapping[KT, VT_co]*)

Update an existing stored record's tags.

Parameters

- **record** – *StorageRecord* to update
- **tags** – New tags

Raises

- `StorageNotFoundError` – If record not found
- `StorageError` – If a libindy error occurs

update_record_value(*record: aries_cloudagent.storage.record.StorageRecord*, *value: str*)

Update an existing stored record's value.

Parameters

- **record** – *StorageRecord* to update
- **value** – The new value

Raises

- `StorageNotFoundError` – If record not found
- `StorageError` – If a libindy error occurs

wallet

Accessor for IndyWallet instance.

class aries_cloudagent.storage.indy.**IndyStorageRecordSearch**(*store: aries_cloudagent.storage.indy.IndyStorage*, *type_filter: str*, *tag_query: Mapping[KT, VT_co]*, *page_size: int = None*)

Bases: *aries_cloudagent.storage.base.BaseStorageRecordSearch*

Represent an active stored records search.

close()

Dispose of the search query.

fetch(*max_count: int*) → Sequence[aries_cloudagent.storage.record.StorageRecord]

Fetch the next list of results from the store.

Parameters **max_count** – Max number of records to return

Returns A list of 'StorageRecord's

Raises StorageSearchError – If the search query has not been opened

handle
Accessor for search handle.

Returns The handle

open()
Start the search query.

opened
Accessor for open state.

Returns True if opened, else False

aries_cloudagent.storage.provider module

Default storage provider classes.

class aries_cloudagent.storage.provider.**StorageProvider**
Bases: *aries_cloudagent.config.base.BaseProvider*

Provider for the default configurable storage classes.

STORAGE_TYPES = {'basic': 'aries_cloudagent.storage.basic.BasicStorage', 'indy': 'ar

provide(*settings: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Create and return the storage instance.

aries_cloudagent.storage.record module

Record instance stored and searchable by BaseStorage implementation.

class aries_cloudagent.storage.record.**StorageRecord**
Bases: *aries_cloudagent.storage.record.StorageRecord*

Storage record class.

1.1.10 aries_cloudagent.tests package

Submodules

aries_cloudagent.tests.test_conductor module

class aries_cloudagent.tests.test_conductor.**Config**
Bases: object

bad_inbound_transports = {'transport.inbound_configs': [['bad', 'host', 80]]}

bad_outbound_transports = {'transport.outbound_configs': ['bad']}

good_inbound_transports = {'transport.inbound_configs': [['http', 'host', 80]]}

good_outbound_transports = {'transport.outbound_configs': ['http']}

test_settings = {}

class aries_cloudagent.tests.test_conductor.**StubContextBuilder**(*settings*)
Bases: aries_cloudagent.config.base_context.ContextBuilder

build() → aries_cloudagent.config.injection_context.InjectionContext
Build the new injection context.

class aries_cloudagent.tests.test_conductor.**TestConductor**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject, *aries_cloudagent.tests.test_conductor.Config*

test_admin()

test_inbound_message_handler()

test_outbound_message_handler()

test_startup()

aries_cloudagent.tests.test_init module

aries_cloudagent.tests.test_stats module

class aries_cloudagent.tests.test_stats.**TestStats**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

test_disable()

test_extract()

test_fn_decorator()

test_method_decorator()

aries_cloudagent.tests.test_task_processor module

class aries_cloudagent.tests.test_task_processor.**RetryTask**(*retries: int*, *result*)
Bases: object

run(*pending: aries_cloudagent.task_processor.PendingTask*)

class aries_cloudagent.tests.test_task_processor.**TestTaskProcessor**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

test_coro()

test_error()

test_retry()

1.1.11 aries_cloudagent.transport package

Subpackages

aries_cloudagent.transport.inbound package

Subpackages

aries_cloudagent.transport.inbound.tests package

Submodules

aries_cloudagent.transport.inbound.tests.test_http_transport module

class aries_cloudagent.transport.inbound.tests.test_http_transport.**TestHttpTransport**(**args*, ***kwarg*

Bases: sphinx.ext.autodoc.importer._MockObject

get_application()

get_transport()

receive_message(*payload*, *scheme*, *single_response=None*)

setUp()

test_send_message()

aries_cloudagent.transport.inbound.tests.test_manager module

class aries_cloudagent.transport.inbound.tests.test_manager.**TestInboundTransportManager**(**ar*, ***k*

Bases: sphinx.ext.autodoc.importer._MockObject

test_register_path()

test_start_stop()

aries_cloudagent.transport.inbound.tests.test_ws_transport module

class aries_cloudagent.transport.inbound.tests.test_ws_transport.**TestWsTransport**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

close_socket()

get_application()

receive_message(*payload*, *scheme*, *socket_id*)

register_socket(*handler*)

setUp()

test_send_message()

Submodules

aries_cloudagent.transport.inbound.base module

Base inbound transport class.

class aries_cloudagent.transport.inbound.base.**BaseInboundTransport**

Bases: abc.ABC

Base inbound transport class.

start() → None

Start listening for on this transport.

stop() → None
Stop listening for on this transport.

class aries_cloudagent.transport.inbound.base.**InboundTransportConfiguration**(*module*, *host*, *port*)

Bases: tuple

host
Alias for field number 1

module
Alias for field number 0

port
Alias for field number 2

exception aries_cloudagent.transport.inbound.base.**InboundTransportRegistrationError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Error in loading an inbound transport.

exception aries_cloudagent.transport.inbound.base.**InboundTransportSetupError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Setup error for an inbound transport.

aries_cloudagent.transport.inbound.http module

Http Transport classes and functions.

class aries_cloudagent.transport.inbound.http.**HttpTransport**(*host: str, port: int, message_router: Coroutine[T_co, T_contra, V_co], register_socket: Coroutine[T_co, T_contra, V_co]*)

Bases: *aries_cloudagent.transport.inbound.base.BaseInboundTransport*

Http Transport class.

inbound_message_handler(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba490e5cc0>*)
Message handler for inbound messages.

Parameters **request** – aiohttp request object

Returns The web response

invite_message_handler(*request: <sphinx.ext.autodoc.importer._MockObject object at 0x7fba490e5cc0>*)
Message handler for invites.

Parameters **request** – aiohttp request object

Returns The web response

make_application() → <sphinx.ext.autodoc.importer._MockObject object at 0x7fba490e5cc0>
Construct the aiohttp application.

scheme
Accessor for this transport's scheme.

start() → None
Start this transport.

Raises InboundTransportSetupError – If there was an error starting the webserver

stop() → None
Stop this transport.

aries_cloudagent.transport.inbound.manager module

Inbound transport manager.

class aries_cloudagent.transport.inbound.manager.**InboundTransportManager**
Bases: object

Inbound transport manager class.

register(*config: aries_cloudagent.transport.inbound.base.InboundTransportConfiguration*, *message_handler*, *register_socket*)
Register transport module.

Parameters

- **module_path** – Path to module
- **host** – The host to register on
- **port** – The port to register on
- **message_handler** – The message handler for incoming messages
- **register_socket** – A coroutine for registering a new socket

register_instance(*transport: aries_cloudagent.transport.inbound.base.BaseInboundTransport*)
Register a new inbound transport instance.

Parameters **transport** – Inbound transport instance to register

start()
Start all registered transports.

stop()
Stop all registered transports.

aries_cloudagent.transport.inbound.ws module

Websockets Transport classes and functions.

class aries_cloudagent.transport.inbound.ws.**WsTransport**(*host: str, port: int, message_router: Coroutine[T_co, T_contra, V_co], register_socket: Coroutine[T_co, T_contra, V_co]*)

Bases: *aries_cloudagent.transport.inbound.base.BaseInboundTransport*

Websockets Transport class.

inbound_message_handler(*request*)
Message handler for inbound messages.

Parameters **request** – aiohttp request object

Returns The web response

make_application() → <sphinx.ext.autodoc.importer._MockObject object at 0x7fba48e26cf8>
Construct the aiohttp application.

scheme
Accessor for this transport's scheme.

start() → None
Start this transport.

Raises InboundTransportSetupError – If there was an error starting the webserver

stop() → None
Stop this transport.

aries_cloudagent.transport.outbound package

Subpackages

aries_cloudagent.transport.outbound.queue package

Subpackages

aries_cloudagent.transport.outbound.queue.tests package

Submodules

aries_cloudagent.transport.outbound.queue.tests.test_basic_queue module

class aries_cloudagent.transport.outbound.queue.tests.test_basic_queue.**TestBasicQueue**(**args, **kwa*

Bases: sphinx.ext.autodoc.importer._MockObject

test_async_iter()

test_enqueue_dequeue()

test_stopped()

aries_cloudagent.transport.outbound.queue.tests.test_basic_queue.**collect**(*queue*, *count=1*)

Submodules

aries_cloudagent.transport.outbound.queue.base module

Abstract outbound queue.

class aries_cloudagent.transport.outbound.queue.base.**BaseOutboundMessageQueue**
Bases: abc.ABC

Abstract outbound queue class.

dequeue(*, *timeout: int = None*)
Dequeue a message.

Returns The dequeued message, or None if a timeout occurs

Raises

- asyncio.CancelledError if the queue has been stopped
- asyncio.TimeoutError if the timeout is reached

enqueue(*message*)
Enqueue a message.

Parameters **message** – The message to add to the end of the queue

Raises asyncio.CancelledError if the queue has been stopped

join()
Wait for the queue to empty.

reset()
Empty the queue and reset the stop event.

stop()
Cancel active iteration of the queue.

task_done()
Indicate that the current task is complete.

aries_cloudagent.transport.outbound.queue.basic module

Basic in memory queue.

class aries_cloudagent.transport.outbound.queue.basic.**BasicOutboundMessageQueue**
Bases: *aries_cloudagent.transport.outbound.queue.base.BaseOutboundMessageQueue*

Basic in memory queue implementation class.

dequeue(*, *timeout: int = None*)
Dequeue a message.

Returns The dequeued message, or None if a timeout occurs

Raises

- asyncio.CancelledError if the queue has been stopped

- asyncio.TimeoutError if the timeout is reached

enqueue(*message*)
Enqueue a message.

Parameters **message** – The message to add to the end of the queue

Raises asyncio.CancelledError if the queue has been stopped

join()
Wait for the queue to empty.

make_queue()
Create the queue instance.

reset()
Empty the queue and reset the stop event.

stop()
Cancel active iteration of the queue.

task_done()
Indicate that the current task is complete.

aries_cloudagent.transport.outbound.tests package

Submodules

aries_cloudagent.transport.outbound.tests.test_http_transport module

class aries_cloudagent.transport.outbound.tests.test_http_transport.**TestHttpTransport**(**args*, ***kwa*

Bases: sphinx.ext.autodoc.importer._MockObject

get_application()
Override the get_app method to return your application.

receive_message(*request*)

setUpAsync()

test_handle_message()

aries_cloudagent.transport.outbound.tests.test_manager module

class aries_cloudagent.transport.outbound.tests.test_manager.**TestOutboundTransportManager**(

Bases: sphinx.ext.autodoc.importer._MockObject

test_register_path()

test_send_message()

aries_cloudagent.transport.outbound.tests.test_ws_transport module

class aries_cloudagent.transport.outbound.tests.test_ws_transport.**TestWsTransport**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

get_application()
Override the get_app method to return your application.

receive_message(*request*)

setUpAsync()

test_handle_message()

Submodules

aries_cloudagent.transport.outbound.base module

Base outbound transport.

class aries_cloudagent.transport.outbound.base.**BaseOutboundTransport**
Bases: abc.ABC

Base outbound transport class.

handle_message(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)
Handle message from queue.

Parameters **message** – *OutboundMessage* to send over transport implementation

start()
Start the transport.

stop()
Shut down the transport.

exception aries_cloudagent.transport.outbound.base.**OutboundTransportRegistrationError**(**args*, *error_code: str = None*, ***kwa*

Bases: *aries_cloudagent.error.BaseError*

Outbound transport registration error.

aries_cloudagent.transport.outbound.http module

Http outbound transport.

class aries_cloudagent.transport.outbound.http.**HttpTransport**
Bases: *aries_cloudagent.transport.outbound.base.BaseOutboundTransport*

Http outbound transport class.

handle_message(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)
Handle message from queue.

Parameters **message** – *OutboundMessage* to send over transport implementation

schemes = ('http', 'https')

start()
Start the transport.

stop()
Stop the transport.

aries_cloudagent.transport.outbound.manager module

Outbound transport manager.

exception aries_cloudagent.transport.outbound.manager.**OutboundDeliveryError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Base exception when a message cannot be delivered via an outbound transport.

class aries_cloudagent.transport.outbound.manager.**OutboundTransportManager**(*queue: aries_cloudagent.trans = None*)

Bases: object

Outbound transport manager class.

dispatch_message(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*, *attempt: int = None*)
Dispatch a message to the relevant transport.

Find a registered transport for the scheme in the uri and use it to send the message.

Parameters **message** – The outbound message to dispatch

get_registered_transport_for_scheme(*scheme: str*)
Find the registered transport for a given scheme.

get_running_transport_for_scheme(*scheme: str*)
Find the running transport for a given scheme.

poll()
Send messages from the queue to the transports.

register(*module_path*)
Register a new outbound transport by module path.

Parameters **module_path** – Module path to register

Raises

- OutboundTransportRegistrationError – If the imported class cannot be located
- OutboundTransportRegistrationError – If the imported class does not specify a schemes attribute
- OutboundTransportRegistrationError – If the scheme has already been registered

register_class(*transport_class: Type[aries_cloudagent.transport.outbound.base.BaseOutboundTransport]*)
Register a new outbound transport class.

Parameters **transport_class** – Transport class to register

Raises

- `OutboundTransportRegistrationError` – If the imported class does not specify a schemes attribute
- `OutboundTransportRegistrationError` – If the scheme has already been registered

send_message(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)
Add a message to the outbound queue.

Parameters **message** – The outbound message to send

start()
Start all transports and feed messages from the queue.

start_transport(*schemes*, *transport_cls*)
Start the transport.

stop(*wait: bool = True*)
Stop all transports.

aries_cloudagent.transport.outbound.ws module

Websockets outbound transport.

class aries_cloudagent.transport.outbound.ws.**WsTransport**
Bases: *aries_cloudagent.transport.outbound.base.BaseOutboundTransport*

Websockets outbound transport class.

handle_message(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)
Handle message from queue.

Parameters **message** – *OutboundMessage* to send over transport implementation

schemes = ('ws', 'wss')

start()
Start the outbound transport.

stop()
Stop the outbound transport.

aries_cloudagent.transport.tests package

Submodules

aries_cloudagent.transport.tests.test_http module

1.1.12 aries_cloudagent.verifier package

Subpackages

aries_cloudagent.verifier.tests package

Submodules

aries_cloudagent.verifier.tests.test_indy module

Submodules

aries_cloudagent.verifier.base module

Base Verifier class.

class aries_cloudagent.verifier.base.**BaseVerifier**
Bases: abc.ABC

Base class for verifier.

aries_cloudagent.verifier.indy module

Indy verifier implementation.

class aries_cloudagent.verifier.indy.**IndyVerifier**(*wallet*)
Bases: *aries_cloudagent.verifier.base.BaseVerifier*

Indy holder class.

verify_presentation(*presentation_request*, *presentation*, *schemas*, *credential_definitions*) → bool
Verify a presentation.

Parameters

- **presentation_request** – Presentation request data
- **presentation** – Presentation data
- **schemas** – Schema data
- **credential_definitions** – credential definition data

1.1.13 aries_cloudagent.wallet package

Abstract and Indy wallet handling.

Subpackages

aries_cloudagent.wallet.tests package

Wallet test suite

Submodules

aries_cloudagent.wallet.tests.test_basic_wallet module

class aries_cloudagent.wallet.tests.test_basic_wallet.**TestBasicWallet**
Bases: object

missing_did = 'YVnYBGTdjZUoQXKQjHV87i'

missing_verkey = 'JAfHCRDH9ZW5E7m4mofjr8cpAHaZdiRQ94it75aXUPK3'

test_create_local_random(*wallet*)

test_create_local_seeded(*wallet*)

test_create_local_with_did(*wallet*)

test_create_retrieve_pairwise(*wallet*)

test_create_signing_key_random(*wallet*)

test_create_signing_key_seeded(*wallet*)

test_did = '55GkHamhTU1ZbTbV2ab9DE'

test_encrypt_decrypt_dids(*wallet*)

test_encrypt_decrypt_keys(*wallet*)

test_local_metadata(*wallet*)

test_local_verkey(*wallet*)

test_message = 'test message'

test_message_bytes = b'test message bytes'

test_metadata = {'meta': True}

test_pack_unpack(*wallet*)

test_pairwise_metadata(*wallet*)

test_seed = 'testseed000000000000000000000001'

test_sign_verify(*wallet*)

test_signature = b'\xd6\x98\x04\x88\xd2-\xc1D\x02\x15\xc9Z\x9bK \x8f\xe0\x8b5\xd0Z$\xe

test_signature_round_trip(*wallet*)

test_signing_key_metadata(*wallet*)

test_target_did = 'GbuDUYXaUZRfHD2jeDuQuP'

test_target_seed = 'testseed000000000000000000000002'

test_target_verkey = '9WCgWKUaAJj3VWxxtzvvMQN3AoFxoBtBDo9ntwJnVVCC'

test_update_metadata = {'meta': False}

test_verkey = '3Dn1SJNPaCXcvvJvSbsFWP2xaCjMom3can8CQNhWrTRx'

aries_cloudagent.wallet.tests.test_basic_wallet.**wallet**()

aries_cloudagent.wallet.tests.test_indy_wallet module

aries_cloudagent.wallet.tests.test_indy_wallet.**basic_wallet**()

aries_cloudagent.wallet.tests.test_indy_wallet.**wallet**()

Submodules

aries_cloudagent.wallet.base module

Wallet base class.

class aries_cloudagent.wallet.base.**BaseWallet**(*config: dict*)
Bases: abc.ABC

Abstract wallet interface.

WALLET_TYPE = None

close()
Close previously-opened wallet, removing it if so configured.

create_local_did(*seed: str = None*, *did: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.DIDInfo
Create and store a new local DID.

Parameters

- **seed** – Optional seed to use for did
- **did** – The DID to use
- **metadata** – Metadata to store with DID

Returns The created *DIDInfo*

create_pairwise(*their_did: str*, *their_verkey: str*, *my_did: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.PairwiseInfo
Create a new pairwise DID for a secure connection.

Parameters

- **their_did** – Their DID
- **their_verkey** – Their verkey
- **my_did** – My DID
- **metadata** – Metadata for relationship

Returns A *PairwiseInfo* instance representing the new relationship

create_public_did(*seed: str = None*, *did: str = None*, *metadata: dict = {}*) → aries_cloudagent.wallet.base.DIDInfo
Create and store a new public DID.

Implicitly flags all other dids as not public.

Parameters

- **seed** – Optional seed to use for did
- **did** – The DID to use
- **metadata** – Metadata to store with DID

Returns The created *DIDInfo*

create_signing_key(*seed: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.KeyInfo
Create a new public/private signing keypair.

Parameters

- **seed** – Optional seed allowing deterministic key creation
- **metadata** – Optional metadata to store with the keypair

Returns A *KeyInfo* representing the new record

created
Check whether the wallet was created on the last open call.

decrypt_message(*enc_message: bytes*, *to_verkey: str*, *use_auth: bool*) -> (<class 'bytes'>, <class 'str'>)
Decrypt a message assembled by auth_crypt or anon_crypt.

Parameters

- **enc_message** – The encrypted message content
- **to_verkey** – The verkey of the recipient. If provided then auth_decrypt is used, otherwise anon_decrypt is used.

Returns

A tuple of the decrypted message content and sender verkey (None for anon_crypt)

encrypt_message(*message: bytes*, *to_verkey: str*, *from_verkey: str = None*) → bytes
Apply auth_crypt or anon_crypt to a message.

Parameters

- **message** – The binary message content
- **to_verkey** – The verkey of the recipient
- **from_verkey** – The verkey of the sender. If provided then auth_crypt is used, otherwise anon_crypt is used.

Returns The encrypted message content

get_local_did(*did: str*) → aries_cloudagent.wallet.base.DIDInfo
Find info for a local DID.

Parameters **did** – The DID to get info for

Returns A *DIDInfo* instance for the DID

get_local_did_for_verkey(*verkey: str*) → aries_cloudagent.wallet.base.DIDInfo
Resolve a local DID from a verkey.

Parameters **verkey** – Verkey to get DID info for

Returns A *DIDInfo* instance for the DID

get_local_dids() → Sequence[aries_cloudagent.wallet.base.DIDInfo]
Get list of defined local DIDs.

Returns A list of *DIDInfo* instances

get_pairwise_for_did(*their_did: str*) → aries_cloudagent.wallet.base.PairwiseInfo
Find info for a pairwise DID.

Parameters **their_did** – The DID representing the relationship

Returns A *PairwiseInfo* instance representing the relationship

get_pairwise_for_verkey(*their_verkey: str*) → aries_cloudagent.wallet.base.PairwiseInfo
Resolve a pairwise DID from a verkey.

Parameters **their_verkey** – The verkey representing the relationship

Returns A *PairwiseInfo* instance representing the relationship

get_pairwise_list() → Sequence[aries_cloudagent.wallet.base.PairwiseInfo]
Get list of defined pairwise DIDs.

Returns A list of *PairwiseInfo* instances for all relationships

get_public_did() → aries_cloudagent.wallet.base.DIDInfo
Retrieve the public did.

Returns The created *DIDInfo*

get_signing_key(*verkey: str*) → aries_cloudagent.wallet.base.KeyInfo
Fetch info for a signing keypair.

Parameters **verkey** – The verification key of the keypair

Returns A *KeyInfo* representing the keypair

handle
Get internal wallet reference.

Returns Defaults to None

name
Accessor for the wallet name.

Returns Defaults to None

open()
Open wallet, removing and/or creating it if so configured.

opened
Check whether wallet is currently open.

Returns Defaults to False

pack_message(*message: str, to_verkeys: Sequence[str], from_verkey: str = None*) → bytes
Pack a message for one or more recipients.

Parameters

- **message** – The message to pack
- **to_verkeys** – The verkeys to pack the message for
- **from_verkey** – The sender verkey

Returns The packed message

replace_local_did_metadata(*did: str*, *metadata: dict*)
Replace the metadata associated with a local DID.

Parameters

- **did** – DID to replace metadata for
- **metadata** – The new metadata

replace_pairwise_metadata(*their_did: str*, *metadata: dict*)
Replace the metadata associated with a pairwise DID.

Parameters

- **their_did** – The did representing the relationship
- **metadata** – The new metadata

replace_signing_key_metadata(*verkey: str*, *metadata: dict*)
Replace the metadata associated with a signing keypair.

Parameters

- **verkey** – The verification key of the keypair
- **metadata** – The new metadata to store

set_public_did(*did: str*) → aries_cloudagent.wallet.base.DIDInfo
Assign the public did.

Returns The created *DIDInfo*

sign_message(*message: bytes*, *from_verkey: str*) → bytes
Sign a message using the private key associated with a given verkey.

Parameters

- **message** – The message to sign
- **from_verkey** – Sign using the private key related to this verkey

Returns The signature

type
Accessor for the wallet type.

Returns Defaults to None

unpack_message(*enc_message: bytes) -> (<class 'str'>*, *<class 'str'>*, *<class 'str'>*)
Unpack a message.

Parameters **enc_message** – The encrypted message

Returns (message, from_verkey, to_verkey)

Return type A tuple

verify_message(*message: bytes*, *signature: bytes*, *from_verkey: str*) → bool
Verify a signature against the public key of the signer.

Parameters

- **message** – The message to verify
- **signature** – The signature to verify
- **from_verkey** – Verkey to use in verification

Returns True if verified, else False

class aries_cloudagent.wallet.base.**DIDInfo**(*did*, *verkey*, *metadata*)
Bases: tuple

did
Alias for field number 0

metadata
Alias for field number 2

verkey
Alias for field number 1

class aries_cloudagent.wallet.base.**KeyInfo**(*verkey*, *metadata*)
Bases: tuple

metadata
Alias for field number 1

verkey
Alias for field number 0

class aries_cloudagent.wallet.base.**PairwiseInfo**(*their_did*, *their_verkey*, *my_did*, *my_verkey*, *metadata*)
Bases: tuple

metadata
Alias for field number 4

my_did
Alias for field number 2

my_verkey
Alias for field number 3

their_did
Alias for field number 0

their_verkey
Alias for field number 1

aries_cloudagent.wallet.basic module

In-memory implementation of BaseWallet interface.

class aries_cloudagent.wallet.basic.**BasicWallet**(*config: dict = None*)
Bases: *aries_cloudagent.wallet.base.BaseWallet*

In-memory wallet implementation.

WALLET_TYPE = 'basic'

close()
Not applicable to in-memory wallet.

create_local_did(*seed: str = None*, *did: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.DIDInfo
Create and store a new local DID.

Parameters

- **seed** – Optional seed to use for did
- **did** – The DID to use
- **metadata** – Metadata to store with DID

Returns A *DIDInfo* instance representing the created DID

Raises WalletDuplicateError – If the DID already exists in the wallet

create_pairwise(*their_did: str*, *their_verkey: str*, *my_did: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.PairwiseInfo
Create a new pairwise DID for a secure connection.

Parameters

- **their_did** – The other party's DID
- **their_verkey** – The other party's verkey

- **my_did** – My DID
- **metadata** – Metadata to store with this relationship

Returns A *PairwiseInfo* object representing the pairwise connection

Raises `WalletDuplicateError` – If the DID already exists in the wallet

create_signing_key(*seed: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.KeyInfo

Create a new public/private signing keypair.

Parameters

- **seed** – Seed to use for signing key
- **metadata** – Optional metadata to store with the keypair

Returns A *KeyInfo* representing the new record

Raises `WalletDuplicateError` – If the resulting verkey already exists in the wallet

created

Check whether the wallet was created on the last open call.

decrypt_message(*enc_message: bytes*, *to_verkey: str*, *use_auth: bool) -> (<class 'bytes'>*, *<class 'str'>*)

Decrypt a message assembled by auth_crypt or anon_crypt.

Parameters

- **message** – The encrypted message content
- **to_verkey** – The verkey of the recipient. If provided then auth_decrypt is used, otherwise anon_decrypt is used.
- **use_auth** – True if you would like to auth_decrypt, False for anon_decrypt

Returns A tuple of the decrypted message content and sender verkey (None for anon_crypt)

encrypt_message(*message: bytes*, *to_verkey: str*, *from_verkey: str = None*) → bytes

Apply auth_crypt or anon_crypt to a message.

Parameters

- **message** – The binary message content
- **to_verkey** – The verkey of the recipient
- **from_verkey** – The verkey of the sender. If provided then auth_crypt is used, otherwise anon_crypt is used.

Returns The encrypted message content

get_local_did(*did: str*) → aries_cloudagent.wallet.base.DIDInfo

Find info for a local DID.

Parameters **did** – The DID to get info for

Returns A *DIDInfo* instance representing the found DID

Raises `WalletNotFoundError` – If the DID is not found

get_local_did_for_verkey(*verkey: str*) → aries_cloudagent.wallet.base.DIDInfo

Resolve a local DID from a verkey.

Parameters **verkey** – The verkey to get the local DID for

Returns A *DIDInfo* instance representing the found DID

Raises `WalletNotFoundError` – If the verkey is not found

get_local_dids() → Sequence[aries_cloudagent.wallet.base.DIDInfo]
Get list of defined local DIDs.

Returns A list of locally stored DIDs as *DIDInfo* instances

get_pairwise_for_did(*their_did: str*) → aries_cloudagent.wallet.base.PairwiseInfo
Find info for a pairwise DID.

Parameters **their_did** – The DID to get a pairwise relationship for

Returns A *PairwiseInfo* instance representing the relationship

Raises `WalletNotFoundError` – If the DID is unknown

get_pairwise_for_verkey(*their_verkey: str*) → aries_cloudagent.wallet.base.PairwiseInfo
Resolve a pairwise DID from a verkey.

Parameters **their_verkey** – The verkey to get a pairwise relationship for

Returns A *PairwiseInfo* instance for the relationship

Raises `WalletNotFoundError` – If the verkey is not found

get_pairwise_list() → Sequence[aries_cloudagent.wallet.base.PairwiseInfo]
Get list of defined pairwise DIDs.

Returns A list of *PairwiseInfo* instances for all pairwise relationships

get_signing_key(*verkey: str*) → aries_cloudagent.wallet.base.KeyInfo
Fetch info for a signing keypair.

Parameters **verkey** – The verification key of the keypair

Returns A *KeyInfo* representing the keypair

Raises `WalletNotFoundError` – if no keypair is associated with the verification key

name
Accessor for the wallet name.

open()
Not applicable to in-memory wallet.

opened
Check whether wallet is currently open.

Returns True

pack_message(*message: str, to_verkeys: Sequence[str], from_verkey: str = None*) → bytes
Pack a message for one or more recipients.

Parameters

- **message** – The message to pack
- **to_verkeys** – List of verkeys to pack for
- **from_verkey** – Sender verkey to pack from

Returns The resulting packed message bytes

replace_local_did_metadata(*did: str*, *metadata: dict*)
Replace metadata for a local DID.

Parameters

- **did** – The DID to replace metadata for
- **metadata** – The new metadata

Raises `WalletNotFoundError` – If the DID doesn't exist

replace_pairwise_metadata(*their_did: str*, *metadata: dict*)
Replace metadata for a pairwise DID.

Parameters

- **their_did** – The DID to replace metadata for
- **metadata** – The new metadata

Raises `WalletNotFoundError` – If the DID is unknown

replace_signing_key_metadata(*verkey: str*, *metadata: dict*)
Replace the metadata associated with a signing keypair.

Parameters

- **verkey** – The verification key of the keypair
- **metadata** – The new metadata to store

Raises `WalletNotFoundError` – if no keypair is associated with the verification key

sign_message(*message: bytes*, *from_verkey: str*) → bytes
Sign a message using the private key associated with a given verkey.

Parameters

- **message** – Message bytes to sign
- **from_verkey** – The verkey to use to sign

Returns A signature

Raises

- `WalletError` – If the message is not provided
- `WalletError` – If the verkey is not provided

unpack_message(*enc_message: bytes) -> (<class 'str'>*, *<class 'str'>*, *<class 'str'>*)
Unpack a message.

Parameters **enc_message** – The packed message bytes

Returns (message, from_verkey, to_verkey)

Return type A tuple

Raises

- `WalletError` – If the message is not provided
- `WalletError` – If there is a problem unpacking the message

verify_message(*message: bytes*, *signature: bytes*, *from_verkey: str*) → bool
Verify a signature against the public key of the signer.

Parameters

- **message** – Message to verify
- **signature** – Signature to verify
- **from_verkey** – Verkey to use in verification

Returns True if verified, else False

Raises

- `WalletError` – If the verkey is not provided
- `WalletError` – If the signature is not provided
- `WalletError` – If the message is not provided

aries_cloudagent.wallet.crypto module

Cryptography functions used by BasicWallet.

class aries_cloudagent.wallet.crypto.**PackMessageSchema**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Packed message schema.

ciphertext
Used by autodoc_mock_imports.

iv
Used by autodoc_mock_imports.

protected
Used by autodoc_mock_imports.

tag
Used by autodoc_mock_imports.

class aries_cloudagent.wallet.crypto.**PackRecipientHeaderSchema**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Packed recipient header schema.

iv
Used by autodoc_mock_imports.

kid
Used by autodoc_mock_imports.

sender
Used by autodoc_mock_imports.

class aries_cloudagent.wallet.crypto.**PackRecipientSchema**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Packed recipient schema.

encrypted_key
Used by autodoc_mock_imports.

header
Used by autodoc_mock_imports.

class aries_cloudagent.wallet.crypto.**PackRecipientsSchema**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Packed recipients schema.

alg
Used by autodoc_mock_imports.

enc
Used by autodoc_mock_imports.

recipients
Used by autodoc_mock_imports.

typ
Used by autodoc_mock_imports.

aries_cloudagent.wallet.crypto.**anon_crypt_message**(*message: bytes*, *to_verkey: bytes*) → bytes

Apply anon_crypt to a binary message.

Parameters

- **message** – The message to encrypt
- **to_verkey** – The verkey to encrypt the message for

Returns The anon encrypted message

aries_cloudagent.wallet.crypto.**anon_decrypt_message**(*enc_message: bytes*, *secret: bytes*) → bytes

Apply anon_decrypt to a binary message.

Parameters

- **enc_message** – The encrypted message
- **secret** – The seed to use

Returns The decrypted message

aries_cloudagent.wallet.crypto.**auth_crypt_message**(*message: bytes*, *to_verkey: bytes*, *from_secret: bytes*) → bytes

Apply auth_crypt to a binary message.

Parameters

- **message** – The message to encrypt
- **to_verkey** – To recipient's verkey
- **from_secret** – The seed to use

Returns The encrypted message

aries_cloudagent.wallet.crypto.**auth_decrypt_message**(*enc_message: bytes*, *secret: bytes*) -> (<class 'bytes'>, <class 'str'>)

Apply auth_decrypt to a binary message.

Parameters

- **enc_message** – The encrypted message
- **secret** – Secret for signing keys

Returns A tuple of (decrypted message, sender verkey)

aries_cloudagent.wallet.crypto.**create_keypair**(*seed: bytes = None*) -> (<class 'bytes'>, <class 'bytes'>)

Create a public and private signing keypair from a seed value.

Parameters **seed** – Seed for keypair

Returns A tuple of (public key, secret key)

aries_cloudagent.wallet.crypto.**decode_pack_message**(*enc_message: bytes, find_key: Callable) -> (<class 'str'>, typing.Union[str, NoneType], <class 'str'>)*

Decode a packed message.

Disassemble and unencrypt a packed message, returning the message content, verification key of the sender (if available), and verification key of the recipient.

Parameters

- **enc_message** – The encrypted message
- **find_key** – Function to retrieve private key

Returns A tuple of (message, sender_vk, recip_vk)

Raises

- ValueError – If the packed message is invalid
- ValueError – If the packed message reipients are invalid
- ValueError – If the pack algorithm is unsupported
- ValueError – If the sender's public key was not provided

aries_cloudagent.wallet.crypto.**decrypt_plaintext**(*ciphertext: bytes, recips_bin: bytes, nonce: bytes, key: bytes*) → str

Decrypt the payload of a packed message.

Parameters

- **ciphertext** –
- **recips_bin** –
- **nonce** –
- **key** –

Returns The decrypted string

aries_cloudagent.wallet.crypto.**encode_pack_message**(*message: str, to_verkeys: Sequence[bytes], from_secret: bytes = None*) → bytes

Assemble a packed message for a set of recipients, optionally including the sender.

Parameters

- **message** – The message to pack
- **to_verkeys** – The verkeys to pack the message for
- **from_secret** – The sender secret

Returns The encoded message

aries_cloudagent.wallet.crypto.**encrypt_plaintext**(*message: str, add_data: bytes, key: bytes) -> (<class 'bytes'>, <class 'bytes'>, <class 'bytes'>)*

Encrypt the payload of a packed message.

Parameters

- **message** – Message to encrypt
- **add_data** –

- **key** – Key used for encryption

Returns A tuple of (ciphertext, nonce, tag)

aries_cloudagent.wallet.crypto.**locate_pack_recipient_key**(*recipients: Sequence[dict], find_key: Callable) -> (<class 'bytes'>, <class 'str'>, <class 'str'>)*

Locate pack recipient key.

Decode the encryption key and sender verification key from a corresponding recipient block, if any is defined.

Parameters

- **recipients** – Recipients to locate
- **find_key** – Function used to find private key

Returns A tuple of (cek, sender_vk, recip_vk_b58)

Raises ValueError – If no corresponding recipient key found

aries_cloudagent.wallet.crypto.**prepare_pack_recipient_keys**(*to_verkeys: Sequence[bytes], from_secret: bytes = None) -> (<class 'str'>, <class 'bytes'>)*

Assemble the recipients block of a packed message.

Parameters

- **to_verkeys** – Verkeys of recipients
- **from_secret** – Secret to use for signing keys

Returns A tuple of (json result, key)

aries_cloudagent.wallet.crypto.**random_seed**() → bytes

Generate a random seed value.

Returns A new random seed

aries_cloudagent.wallet.crypto.**seed_to_did**(*seed: str*) → str

Derive a did from a seed value.

Parameters **seed** – The seed to derive

Returns The did derived from the seed

aries_cloudagent.wallet.crypto.**sign_message**(*message: bytes*, *secret: bytes*) → bytes

Sign a message using a private signing key.

Parameters

- **message** – The message to sign
- **secret** – The private signing key

Returns The signature

aries_cloudagent.wallet.crypto.**validate_seed**(*seed: (<class 'str'>, <class 'bytes'>)*) → bytes

Convert a seed parameter to standard format and check length.

Parameters **seed** – The seed to validate

Returns The validated and encoded seed

aries_cloudagent.wallet.crypto.**verify_signed_message**(*signed: bytes*, *verkey: bytes*) → bool

Verify a signed message according to a public verification key.

Parameters

- **signed** – The signed message
- **verkey** – The verkey to use in verification

Returns True if verified, else False

aries_cloudagent.wallet.error module

Wallet-related exceptions.

exception aries_cloudagent.wallet.error.**WalletDuplicateError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.wallet.error.WalletError*

Duplicate record exception.

exception aries_cloudagent.wallet.error.**WalletError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

General wallet exception.

exception aries_cloudagent.wallet.error.**WalletNotFoundError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.wallet.error.WalletError*

Record not found exception.

aries_cloudagent.wallet.indy module

Indy implementation of BaseWallet interface.

class aries_cloudagent.wallet.indy.**IndyWallet**(*config: dict = None*)

Bases: *aries_cloudagent.wallet.base.BaseWallet*

Indy wallet implementation.

DEFAULT_FRESHNESS = 0

DEFAULT_KEY = ''

DEFAULT_NAME = 'default'

DEFAULT_STORAGE_TYPE = None

WALLET_TYPE = 'indy'

close()

Close previously-opened wallet, removing it if so configured.

create(*replace: bool = False*)

Create a new wallet.

Parameters **replace** – Removes the old wallet if True

Raises

- `WalletError` – If there was a problem removing the wallet
- `WalletError` – IF there was a libindy error

create_local_did(*seed: str = None*, *did: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.DIDInfo

Create and store a new local DID.

Parameters

- **seed** – Optional seed to use for did
- **did** – The DID to use
- **metadata** – Metadata to store with DID

Returns A *DIDInfo* instance representing the created DID

Raises

- `WalletDuplicateError` – If the DID already exists in the wallet
- `WalletError` – If there is a libindy error

create_pairwise(*their_did: str*, *their_verkey: str*, *my_did: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.PairwiseInfo

Create a new pairwise DID for a secure connection.

Parameters

- **their_did** – The other party's DID
- **their_verkey** – The other party's verkey
- **my_did** – My DID
- **metadata** – Metadata to store with this relationship

Returns A *PairwiseInfo* object representing the pairwise connection

Raises

- `WalletError` – If there is a libindy error
- `WalletDuplicateError` – If the DID already exists in the wallet

create_signing_key(*seed: str = None*, *metadata: dict = None*) → aries_cloudagent.wallet.base.KeyInfo

Create a new public/private signing keypair.

Parameters

- **seed** – Seed for key
- **metadata** – Optional metadata to store with the keypair

Returns A *KeyInfo* representing the new record

Raises

- `WalletDuplicateError` – If the resulting verkey already exists in the wallet
- `WalletError` – If there is a libindy error

created

Check whether the wallet was created on the last open call.

decrypt_message(*enc_message: bytes*, *to_verkey: str*, *use_auth: bool) -> (<class 'bytes'>, <class 'str'>)*
Decrypt a message assembled by auth_crypt or anon_crypt.

Parameters

- **message** – The encrypted message content
- **to_verkey** – The verkey of the recipient. If provided then auth_decrypt is used, otherwise anon_decrypt is used.
- **use_auth** – True if you would like to auth_decrypt, False for anon_decrypt

Returns A tuple of the decrypted message content and sender verkey (None for anon_crypt)

Raises `WalletError` – If a libindy error occurs

encrypt_message(*message: bytes*, *to_verkey: str*, *from_verkey: str = None*) → bytes
Apply auth_crypt or anon_crypt to a message.

Parameters

- **message** – The binary message content
- **to_verkey** – The verkey of the recipient
- **from_verkey** – The verkey of the sender. If provided then auth_crypt is used, otherwise anon_crypt is used.

Returns The encrypted message content

Raises `WalletError` – If a libindy error occurs

get_local_did(*did: str*) → aries_cloudagent.wallet.base.DIDInfo
Find info for a local DID.

Parameters **did** – The DID to get info for

Returns A *DIDInfo* instance representing the found DID

Raises

- `WalletNotFoundError` – If the DID is not found
- `WalletError` – If there is a libindy error

get_local_did_for_verkey(*verkey: str*) → aries_cloudagent.wallet.base.DIDInfo
Resolve a local DID from a verkey.

Parameters **verkey** – The verkey to get the local DID for

Returns A *DIDInfo* instance representing the found DID

Raises `WalletNotFoundError` – If the verkey is not found

get_local_dids() → Sequence[aries_cloudagent.wallet.base.DIDInfo]
Get list of defined local DIDs.

Returns A list of locally stored DIDs as *DIDInfo* instances

get_pairwise_for_did(*their_did: str*) → aries_cloudagent.wallet.base.PairwiseInfo
Find info for a pairwise DID.

Parameters **their_did** – The DID to get a pairwise relationship for

Returns A *PairwiseInfo* instance representing the relationship

Raises

- WalletNotFoundError – If no pairwise DID defined for target
- WalletNotFoundError – If no pairwise DID defined for target

get_pairwise_for_verkey(*their_verkey: str*) → aries_cloudagent.wallet.base.PairwiseInfo
Resolve a pairwise DID from a verkey.

Parameters **their_verkey** – The verkey to get a pairwise relationship for

Returns A *PairwiseInfo* instance for the relationship

Raises WalletNotFoundError – If no pairwise DID is defined for verkey

get_pairwise_list() → Sequence[aries_cloudagent.wallet.base.PairwiseInfo]
Get list of defined pairwise DIDs.

Returns A list of *PairwiseInfo* instances for all pairwise relationships

get_signing_key(*verkey: str*) → aries_cloudagent.wallet.base.KeyInfo
Fetch info for a signing keypair.

Parameters **verkey** – The verification key of the keypair

Returns A *KeyInfo* representing the keypair

Raises

- WalletNotFoundError – If no keypair is associated with the verification key
- WalletError – If there is a libindy error

handle
Get internal wallet reference.

Returns A handle to the wallet

master_secret_id
Accessor for the master secret id.

Returns The master secret id

name
Accessor for the wallet name.

Returns The wallet name

open()
Open wallet, removing and/or creating it if so configured.

Raises

- WalletError – If wallet not found after creation
- WalletNotFoundError – If the wallet is not found
- WalletError – If the wallet is already open
- WalletError – If there is a libindy error

opened
Check whether wallet is currently open.

Returns True if open, else False

pack_message(*message: str, to_verkeys: Sequence[str], from_verkey: str = None*) → bytes
Pack a message for one or more recipients.

Parameters

- **message** – The message to pack
- **to_verkeys** – List of verkeys to pack for
- **from_verkey** – Sender verkey to pack from

Returns The resulting packed message bytes

Raises

- `WalletError` – If no message is provided
- `WalletError` – If a libindy error occurs

remove()
Remove an existing wallet.

Raises

- `WalletNotFoundError` – If the wallet could not be found
- `WalletError` – If there was an libindy error

replace_local_did_metadata(*did: str*, *metadata: dict*)
Replace metadata for a local DID.

Parameters

- **did** – The DID to replace metadata for
- **metadata** – The new metadata

replace_pairwise_metadata(*their_did: str*, *metadata: dict*)
Replace metadata for a pairwise DID.

Parameters

- **their_did** – The DID to replace metadata for
- **metadata** – The new metadata

replace_signing_key_metadata(*verkey: str*, *metadata: dict*)
Replace the metadata associated with a signing keypair.

Parameters

- **verkey** – The verification key of the keypair
- **metadata** – The new metadata to store

Raises `WalletNotFoundError` – if no keypair is associated with the verification key

sign_message(*message: bytes*, *from_verkey: str*) → bytes
Sign a message using the private key associated with a given verkey.

Parameters

- **message** – Message bytes to sign
- **from_verkey** – The verkey to use to sign

Returns A signature

Raises

- `WalletError` – If the message is not provided
- `WalletError` – If the verkey is not provided
- `WalletError` – If a libindy error occurs

unpack_message(*enc_message: bytes) -> (<class 'str'>, <class 'str'>, <class 'str'>*)
Unpack a message.

Parameters **enc_message** – The packed message bytes

Returns (message, from_verkey, to_verkey)

Return type A tuple

Raises

- WalletError – If the message is not provided
- WalletError – If a libindy error occurs

verify_message(*message: bytes*, *signature: bytes*, *from_verkey: str*) → bool
Verify a signature against the public key of the signer.

Parameters

- **message** – Message to verify
- **signature** – Signature to verify
- **from_verkey** – Verkey to use in verification

Returns True if verified, else False

Raises

- WalletError – If the verkey is not provided
- WalletError – If the signature is not provided
- WalletError – If the message is not provided
- WalletError – If a libindy error occurs

aries_cloudagent.wallet.provider module

Default wallet provider classes.

class aries_cloudagent.wallet.provider.**WalletProvider**
Bases: *aries_cloudagent.config.base.BaseProvider*

Provider for the default configurable wallet classes.

WALLET_TYPES = {'basic': 'aries_cloudagent.wallet.basic.BasicWallet', 'indy': 'aries

provide(*settings: aries_cloudagent.config.base.BaseSettings*, *injector: aries_cloudagent.config.base.BaseInjector*)
Create and open the wallet instance.

aries_cloudagent.wallet.util module

Wallet utility functions.

aries_cloudagent.wallet.util.**b58_to_bytes**(*val: str*) → bytes
Convert a base 58 string to bytes.

aries_cloudagent.wallet.util.**b64_to_bytes**(*val: str*, *urlsafe=False*) → bytes
Convert a base 64 string to bytes.

aries_cloudagent.wallet.util.**bytes_to_b58**(*val: bytes*) → str
Convert a byte string to base 58.

aries_cloudagent.wallet.util.**bytes_to_b64**(*val: bytes*, *urlsafe=False*) → str
Convert a byte string to base 64.

1.2 Submodules

1.3 aries_cloudagent.classloader module

The classloader provides utilties to dynamically load classes and modules.

class aries_cloudagent.classloader.**ClassLoader**(*base_path*, *super_class*)
Bases: object

Class used to load classes from modules dynamically.

load(*module_path*, *load_relative=False*)
Load module by module path.

Parameters

- **module_path** – Dotted path to module
- **load_relative** – Should the method check in the
- **base path for relative import**(*configured*) –

Returns The loaded class

Raises

- *ModuleLoadError* – If there is an error loading the class
- *ClassNotFoundError* – If there is no class to load at specified path

classmethod load_class(*class_name: str*, *default_module: str = None*)
Resolve a complete class path (ie. typing.Dict) to the class itself.

Parameters

- **class_name** – Class name
- **default_module** – (Default value = None)

Returns The resolved class

Raises

- *ClassNotFoundError* – If the class could not be resolved at path
- *ModuleLoadError* – If there was an error loading the module

classmethod load_module(*mod_path: str*, *default_module: str = None*)
Resolve a complete class path (ie. typing.Dict) to the class itself.

Parameters

- **class_name** – Class name
- **default_module** – (Default value = None)

Returns The resolved class

Raises

- *ClassNotFoundError* – If the class could not be resolved at path

- *ModuleLoadError* – If there was an error loading the module

exception aries_cloudagent.classloader.**ClassNotFoundError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Class not found error.

exception aries_cloudagent.classloader.**ModuleLoadError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Module load error.

1.4 aries_cloudagent.conductor module

The Conductor.

The conductor is responsible for coordinating messages that are received over the network, communicating with the ledger, passing messages to handlers, instantiating concrete implementations of required modules and storing data in the wallet.

class aries_cloudagent.conductor.**Conductor**(*context_builder: aries_cloudagent.config.base_context.ContextBuilder*)

Bases: object

Conductor class.

Class responsible for initializing concrete implementations of our require interfaces and routing inbound and outbound message data.

inbound_message_router(*message_body: Union[str, bytes]*, *transport_type: str = None*, *socket_id: str = None*, *single_response: _asyncio.Future = None*) → _asyncio.Future

Route inbound messages.

Parameters

- **message_body** – Body of the incoming message
- **transport_type** – Type of transport this message came from
- **socket_id** – The identifier of the incoming socket connection
- **single_response** – A future to contain the first direct response message

outbound_message_router(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*, *context: aries_cloudagent.config.injection_context.InjectionContext = None*) → None

Route an outbound message.

Parameters

- **message** – An outbound message to be sent
- **context** – Optional request context

prepare_outbound_message(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*, *context: aries_cloudagent.config.injection_context.InjectionContext = None*)

Prepare a response message for transmission.

Parameters

- **message** – An outbound message to be sent
- **context** – Optional request context

register_socket(*, *handler: Coroutine[T_co, T_contra, V_co] = None*, *single_response: _asyncio.Future = None*) → aries_cloudagent.messaging.socket.SocketRef
Register a new duplex connection.

setup()
Initialize the global request context.

start() → None
Start the agent.

stop(*timeout=0.1*)
Stop the agent.

1.5 aries_cloudagent.defaults module

Sane defaults for known message definitions.

aries_cloudagent.defaults.**default_protocol_registry**() → aries_cloudagent.messaging.protocol_registry.ProtocolRe
Protocol registry for default message types.

1.6 aries_cloudagent.dispatcher module

The Dispatcher.

The dispatcher is responsible for coordinating data flow between handlers, providing lifecycle hook callbacks storing state for message threads, etc.

class aries_cloudagent.dispatcher.**Dispatcher**(*context: aries_cloudagent.config.injection_context.InjectionContext*
Bases: object

Dispatcher class.

Class responsible for dispatching messages to message handlers and responding to other agents.

dispatch(*parsed_msg: dict, delivery: aries_cloudagent.messaging.message_delivery.MessageDelivery, connection: aries_cloudagent.messaging.connections.models.connection_record.ConnectionRecord, send: Coroutine[T_co, T_contra, V_co]*) → _asyncio.Future
Configure responder and dispatch message context to message handler.

Parameters

- **parsed_msg** – The parsed message body
- **delivery** – The incoming message delivery metadata
- **connection** – The related connection record, if any
- **send** – Function to send outbound messages

Returns The response from the handler

make_message(*parsed_msg: dict*) → aries_cloudagent.messaging.agent_message.AgentMessage
Deserialize a message dict into the appropriate message instance.

Given a dict describing a message, this method returns an instance of the related message class.

Parameters **parsed_msg** – The parsed message

Returns An instance of the corresponding message class for this message

Raises

- MessageParseError – If the message doesn't specify @type
- MessageParseError – If there is no message class registered to handle
- the given type

class aries_cloudagent.dispatcher.**DispatcherResponder**(*send: Coroutine[T_co, T_contra, V_co], context: aries_cloudagent.messaging.request_context.RequestC **kwargs*)

Bases: *aries_cloudagent.messaging.responder.BaseResponder*

Handle outgoing messages from message handlers.

create_outbound(*message: Union[aries_cloudagent.messaging.agent_message.AgentMessage, str, bytes], **kwargs*) → aries_cloudagent.messaging.outbound_message.OutboundMessage

Create an OutboundMessage from a message body.

send_outbound(*message: aries_cloudagent.messaging.outbound_message.OutboundMessage*)

Send outbound message.

Parameters **message** – The *OutboundMessage* to be sent

send_webhook(*topic: str*, *payload: dict*)

Dispatch a webhook.

Parameters

- **topic** – the webhook topic identifier
- **payload** – the webhook payload value

1.7 aries_cloudagent.error module

Common exception classes.

exception aries_cloudagent.error.**BaseError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: Exception

Generic exception class which other exceptions should inherit from.

error_code = None

message

Accessor for the error message.

exception aries_cloudagent.error.**StartupError**(**args*, *error_code: str = None*, ***kwargs*)

Bases: *aries_cloudagent.error.BaseError*

Error raised when there is a problem starting the conductor.

1.8 aries_cloudagent.postgres module

Utility for loading Postgres wallet plug-in.

aries_cloudagent.postgres.**file_ext**()
Determine file extension based on platform.

aries_cloudagent.postgres.**load_postgres_plugin**()
Load postgres dll and configure postgres wallet.

1.9 aries_cloudagent.stats module

Classes for tracking performance and timing.

class aries_cloudagent.stats.**Collector**(*enabled: bool = True*)
Bases: object

Collector for a set of statistics.

enabled
Accessor for the collector's enabled property.

extract(*groups: Sequence[str] = None*) → dict
Extract statistics for a specific set of groups.

log(*name: str*, *duration: float*)
Log an entry in the statistics if the collector is enabled.

mark(**names*)
Make a custom decorator function for adding to the set of groups.

reset()
Reset the collector's statistics.

results
Accessor for the current set of collected statistics.

timer(**groups*)
Create a new timer attached to this collector.

wrap(*obj*, *prop_name: Union[str, Sequence[str]]*, *groups: Sequence[str] = None*, ***, *ignore_missing: bool = False*)
Wrap a method on a class or class instance.

wrap_coro(*fn*, *groups: Sequence[str]*)
Wrap a coroutine instance to collect timing statistics on execution.

wrap_fn(*fn*, *groups: Sequence[str]*)
Wrap a function instance to collect timing statistics on execution.

class aries_cloudagent.stats.**Stats**
Bases: object

A collection of statistics.

extract(*names: Sequence[str] = None*) → dict
Summarize the stats in a dictionary.

log(*name: str*, *duration: float*)
Log an entry in the stats.

class aries_cloudagent.stats.**Timer**(*collector: aries_cloudagent.stats.Collector*, *groups: Sequence[str]*)
Bases: object

Timer instance for a running task.

classmethod now()
Fetch a standard timer value.

1.10 aries_cloudagent.task_processor module

Classes for managing a limited set of concurrent tasks.

class aries_cloudagent.task_processor.**PendingTask**(*ident, fn: Callable[[PendingTask], Awaitable[T_co]], retries: int = None, retry_delay: float = None*)
Bases: object

Class for tracking pending tasks.

cancel()
Cancel the running task.

done()
Check if the task is done.

exception()
Get the exception raised by the task, if any.

result()
Get the result of the task.

class aries_cloudagent.task_processor.**TaskProcessor**(**, max_pending: int = 10*)
Bases: object

Class for managing a limited set of concurrent tasks.

done()
Check if the processor has any pending tasks.

ready()
Check if the processor is ready.

run_retry(*fn: Callable[[aries_cloudagent.task_processor.PendingTask], Awaitable[T_co]], *, ident=None, retries: int = 5, retry_delay: float = 10.0, when_ready: bool = True*) → aries_cloudagent.task_processor.PendingTask
Process a task and track the result.

run_task(*task: Awaitable[T_co], *, ident=None, when_ready: bool = True*) → aries_cloudagent.task_processor.PendingTask
Run a single coroutine with no retries.

wait_done()
Wait for all pending tasks to complete.

wait_ready()
Wait for the processor to be ready for more tasks.

aries_cloudagent.task_processor.**delay_task**(*delay: float, task: Awaitable[T_co]*)
Wait a given amount of time before executing an awaitable.

1.11 aries_cloudagent.version module

Library version information.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W