Basic Concepts

Before diving into the API, it's important to understand the fundamental data types of IOTA.

todo: Link to IOTA docs

PyOTA Types

PyOTA defines a few types that will make it easy for you to model objects like Transactions and Bundles in your own code.

TryteString

```
from iota import TryteString

trytes_1 = TryteString(b'RBTC9D9DCDQAEASBYBCCKBFA')
trytes_2 = TryteString(b'LH9GYEMHCF9GWHZFEELHVFOEOHNEEEWHZFUD')

if trytes_1 != trytes_2:
  trytes_combined = trytes_1 + trytes_2

index = {
  trytes_1: 42,
  trytes_2: 86,
}
```

A `TryteString` is an ASCII representation of a sequence of trytes. In many respects, it is similar to a Python `bytes` object (which is an ASCII representation of a sequence of bytes).

In fact, the two objects behave very similarly; they support concatenation, comparison, can be used as dict keys, etc.

However, unlike `bytes`, a `TryteString` can only contain uppercase letters and the number 9 (as a regular expression: `^[A-Z9]*$`).

As you go through the API documentation, you will see many references to `TryteString` and its subclasses:

- `Fragment`: A signature or message fragment inside a transaction. Fragments are always 2187 trytes long.
- `Hash`: An object identifier. Hashes are always 81 trytes long. There are many different types of hashes:
- `Address`: Identifies an address on the Tangle.
- `BundleHash`: Identifies a bundle on the Tangle.
- `TransactionHash`: Identifies a transaction on the Tangle.
- `Seed`: A TryteString that is used for crypto functions such as generating addresses, signing inputs, etc. Seeds can be any length, but 81 trytes offers the best security.
- `Tag`: A tag used to classify a transaction. Tags are always 27 trytes long.
- `TransactionTrytes`: A TryteString representation of a transaction on the Tangle. `TransactionTrytes` are always 2673 trytes long.

Encoding

```
from iota import TryteString

message_trytes = TryteString.from_unicode('Hello, IOTA!')
```

To encode character data into trytes, use the `TryteString.from_unicode` method.

You can also convert a tryte sequence into characters using `TryteString.decode`. Note that not every tryte sequence can be converted; garbage in, garbage out!

```
from iota import TryteString

trytes = TryteString(b'RBTC9D9DCDQAEASBYBCCKBFA')
message = trytes.decode()
```

Note

PyOTA also supports encoding non-ASCII characters, but this functionality is **experimental** and has not yet been evaluated by the IOTA community!

Until this feature has been standardized, it is recommended that you only use ASCII characters when generating `TryteString` objects from character strings.

Transaction Types

PyOTA defines two different types used to represent transactions:

Transaction

```python
from iota import Address, ProposedTransaction, Tag, Transaction

txn_1 =\
  Transaction.from_tryte_string(
    b'GYPRVHBEZOOFXSHQBLCYW9ICTCISLHDBNMMVYD9JJHQMPQCTIQAQTJNNNJ9IDXLRCC'
    b'OYOXYPCLR9PBEY9ORZIEPPDNTI9CQWYZUOTAVBXPSBOFEQAPFLWXSWUIUSJMSJIIIZ'
    b'WIKIRH9GCOEVZFKNXEVCUCIIWZQCQEUVRZOCMEL9AMGXJNMLJCIA9UWGRPPHCEOPTS'
    b'VPKPPPCMQXYBHMSODTWUOABPKWFFFQJHCBVYXLHEWPD9YUDFTGNCYAKQKVEZYRBQRB'
    b'XIAUX9SVEDUKGMTWQIYXRGSWYRK9SRONVGTW9YGHSZRIXWGPCCUCDRMAXBPDFVHSRY'
    b'WHGB9DQSQFQKSNICGPIPTRZINYRXQAFSWSEWIFRMSBMGTNYPRWFSOIIWWT9IDSELM9'
    b'JUOOWFNCCSHUSMGNROBFJX9JQ9XT9PKEGQYQAWAFPRVRRVQPUQBHLSNTEFCDKBWRCD'
    b'X9EYOBB9KPMTLNNQLADBDLZPRVBCKVCYQEOLARJYAGTBFR9QLPKZBOYWZQOVKCVYRG'
    b'YI9ZEFIQRKYXLJBZJDBJDJVQZCGYQMROVHNDBLGNLQODPUXFNTADDVYNZJUVPGB9LV'
    b'PJIYLAPBOEHPMRWUIAJXVQOEM9ROEYUOTNLXVVQEYRQWDTQGDLEYFIYNDPRAIXOZEB'
    b'CS9P99AZTQQLKEILEVXMSHBIDHLXKUOMMNFKPYHONKEYDCHMUNTTNRYVMMEYHPGASP'
    b'ZXASKRUPWQSHDMU9VPS99ZZ9SJJYFUJFFMFORBYDILBXCAVJDPDFHTTTIYOVGLRDYR'
    b'TKHXJORJVYRPTDH9ZCPZ9ZADXZFRSFPIQKWLBRNTWJHXTOAUOL9FVGTUMMPYGYICJD'
    b'XMOESEVDJWLMCVTJLPIEKBE9JTHDQWV9MRMEWFLPWGJFLUXI9BXPSVWCMUWLZSEWHB'
    b'DZKXOLYNOZAPOYLQVZAQMOHGTTQEUAOVKVRRGAHNGPUEKHFVPVCOYSJAWHZU9DRROH'
    b'BETBAFTATVAUGOEGCAYUXACLSSHHVYDHMDGJP9AUCLWLNTFEVGQGHQXSKEMVOVSKQE'
    b'EWHWZUDTYOBGCURRZSJZLFVQQAAYQO9TRLFFN9HTDQXBSPPJYXMNGLLBHOMNVXNOWE'
    b'IDMJVCLLDFHBDONQJCJVLBLCSMDOUQCKKCQJMGTSTHBXPXAMLMSXRIPUBMBAWBFNLH'
    b'LUJTRJLDERLZFUBUSMF999XNHLEEXEENQJNOFFPNPQ9PQICHSATPLZVMVIWLRTKYPI'
    b'XNFGYWOJSQDAXGFHKZPFLPXQEHCYEAGTIWIJEZTAVLNUMAFWGGLXMBNUQTOFCNLJTC'
    b'DMWVVZGVBSEBCPFSM99FLOIDTCLUGPSEDLOKZUAEVBLWNMODGZBWOVQT9DPFOTSKRA'
    b'BQAVOQ9RXWBMAKFYNDCZOJGTCIDMQSQQSODKDXTPFLNOKSIZEOY9HFUTLQRXQMEPGO'
    b'XQGLLPNSXAUCYPGZMNWMQWSWCKAQYKXJTWINSGPPZG9HLDLEAWUWEVCTVRCBDFOXKU'
    b'ROXH9HXXAXVPEJFRSLOGRVGYZASTEBAQNXJJROCYRTDPYFUIQJVDHAKEG9YACV9HCP'
    b'JUEUKOYFNWDXCCJBIFQKYOXGRDHVTHEQUMHO999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999999999999999999999999999999999999'
    b'999999999999RKWEEVD99A99999999A99999999NFDPEEZCWVYLKZGSLCQNOFUSENI'
    b'XRHWWTZFBXMPSQHEDFWZULBZFEOMNLRNIDQKDNNIELAOXOVMYEI9PGTKORV9IKTJZQ'
    b'UBQAWTKBKZ9NEZHBFIMCLV9TTNJNQZUIJDFPTTCTKBJRHAITVSKUCUEMD9M9SQJ999'
    b'999TKORV9IKTJZQUBQAWTKBKZ9NEZHBFIMCLV9TTNJNQZUIJDFPTTCTKBJRHAITVSK'
    b'UCUEMD9M9SQJ999999999999999999999999999999999999999999999999999999'
    b'999999999999999999999999999999999'
  )
```

`Transaction` is a transaction that has been loaded from the Tangle.

Generally, you will never need to create `Transaction` objects; the API will build them for you, as the result of various API methods.

Each `Transaction` has the following attributes:

- `address: Address`: The address associated with this transaction. Depending on the transaction's `value`, this address may be a sender or a recipient.
- `attachment_timestamp: int`: Estimated epoch time of the attachment to the tangle.
- `attachment_time_lower_bound: int`: The lowest possible epoch time of the attachment to the tangle.
- `attachment_time_upper_bound: int`: The highest possible epoch time of the attachment to the tangle.
- `branch_transaction_hash: TransactionHash`: An unrelated transaction that this transaction "approves". Refer to the Basic Concepts section for more information.
- `bundle_hash: BundleHash`: The bundle hash, used to identify transactions that are part of the same bundle. This value is generated by taking a hash of the metadata from all transactions in the bundle.
- `current_index: int`: The transaction's position in the bundle.
- If the `current_index` value is 0, then this is the "tail transaction".
- If it is equal to `last_index`, then this is the "head transaction".
- `hash: TransactionHash`: The transaction hash, used to uniquely identify the transaction on the Tangle. This value is generated by taking a hash of the raw transaction trits.
- `is_confirmed: Optional[bool]`: Whether this transaction has been "confirmed". Refer to the Basic Concepts section for more information.
- `last_index: int`: The index of the final transaction in the bundle. This value is attached to every transaction to make it easier to traverse and verify bundles.
- `legacy_tag: Tag`: A short message attached to the transaction. Deprecated, use `tag` instead.
- `nonce: Hash`: This is the product of the PoW process.
- `signature_message_fragment: Fragment`: Additional data attached to the transaction:
- If `value < 0`, this value contains a fragment of the cryptographic signature authorizing the spending of the IOTAs.
- If `value > 0`, this value is an (optional) string message attached to the

transaction.

- If `value = 0`, this value could be either a signature or message fragment, depending on the previous transaction.
- `tag: Tag`: Used to classify the transaction. Many transactions have empty tags (`Tag(b'999999999999999999999999999')`).
- `timestamp: int`: Unix timestamp when the transaction was created. Note that devices can specify any timestamp when creating transactions, so this value is not safe to use for security measures (such as resolving double-spends).
- `trunk_transaction_hash: TransactionHash`: The transaction hash of the next transaction in the bundle. If this transaction is the head transaction, its `trunk_transaction_hash` will be pseudo-randomly selected, similarly to `branch_transaction_hash`.
- `value: int`: The number of IOTAs being transferred in this transaction:
- If this value is negative, then the `address` is spending IOTAs.
- If it is positive, then the `address` is receiving IOTAs.
- If it is zero, then this transaction is being used to carry metadata (such as a signature fragment or a message) instead of transferring IOTAs.

ProposedTransaction

`ProposedTransaction` is a transaction that was created locally and hasn't been broadcast yet.

```
txn_2 =\
  ProposedTransaction(
    address =
      Address(
        b'TESTVALUE9DONTUSEINPRODUCTION99999XE9IVG'
        b'EFNDOCQCMERGUATCIEGGOHPHGFIAQEZGNHQ9W99CH'
      ),

    message = TryteString.from_unicode('thx fur cheezburgers'),
    tag     = Tag(b'KITTEHS'),
    value   = 42,
  )
```

This type is useful when creating new transactions to broadcast to the Tangle. Note that creating a `ProposedTransaction` requires only a small subset of the attributes needed to create a `Transaction` object.

To create a `ProposedTransaction`, specify the following values:

- `address: Address`: The address associated with the transaction. Note that each transaction references exactly one address; in order to transfer IOTAs from one address to another, you must create at least two transactions: One to deduct the IOTAs from the sender's balance, and one to add the IOTAs to the recipient's balance.
- `message: Optional[TryteString]`: Optional trytes to attach to the transaction. This could be any value (character strings, binary data, or raw trytes), as long as it's converted to a `TryteString` first.
- `tag: Optional[Tag]`: Optional tag to classify this transaction. Each transaction may have exactly one tag, and the tag is limited to 27 trytes.
- `value: int`: The number of IOTAs being transferred in this transaction. This value can be 0; for example, to send a message without spending any IOTAs.

Bundle Types

As with transactions, PyOTA defines two bundle types.

Bundle

```
from iota import Bundle

bundle = Bundle.from_tryte_strings([
  b'GYPRVHBEZOOFXSHQBLCYW9ICTCISLHDBNMMVYD9JJHQMPQCTIQAQTJNNNJ9IDXLRCC...',
  b'OYOXYPCLR9PBEY9ORZIEPPDNTI9CQWYZUOTAVBXPSBOFEQAPFLWXSWUIUSJMSJIIIZ...',
  # etc.
])
```

`Bundle` represents a bundle of transactions published on the Tangle. It is intended to be a read-only object, allowing you to inspect the transactions and bundle metadata.

Each bundle has the following attributes:

- `hash: BundleHash`: The hash of this bundle. This value is generated by taking a hash of the metadata from all transactions in the bundle.
- `is_confirmed: Optional[bool]`: Whether the transactions in this bundle have been confirmed. Refer to the Basic Concepts section for more information.
- `tail_transaction: Optional[Transaction]`: The bundle's tail transaction.
- `transactions: List[Transaction]`: The transactions associated with this bundle.

ProposedBundle

```python
from iota import Address, ProposedBundle, ProposedTransaction
from iota.crypto.signing import KeyGenerator

bundle = ProposedBundle()

bundle.add_transaction(ProposedTransaction(...))
bundle.add_transaction(ProposedTransaction(...))
bundle.add_transaction(ProposedTransaction(...))

bundle.add_inputs([
  Address(
    address =
      b'TESTVALUE9DONTUSEINPRODUCTION99999HAA9UA'
      b'MHCGKEUGYFUBIARAXBFASGLCHCBEVGTBDCSAEBTBM',

    balance   = 86,
    key_index = 0,
  ),
])

bundle.send_unspent_inputs_to(
  Address(
    b'TESTVALUE9DONTUSEINPRODUCTION99999D99HEA'
    b'M9XADCPFJDFANCIHR9OBDHTAGGE9TGCI9EO9ZCRBN'
  ),
)

bundle.finalize()
bundle.sign_inputs(KeyGenerator(b'SEED9GOES9HERE'))
```

Note

This section contains information about how PyOTA works “under the hood”.

> The `prepare_transfer` API method encapsulates this functionality for you; it is not necessary to understand how `ProposedBundle` works in order to use PyOTA.

`ProposedBundle` provides a convenient interface for creating new bundles, listed in the order that they should be invoked:

- `add_transaction: (ProposedTransaction) -> None`: Adds a transaction to the bundle. If necessary, it may split the transaction into multiple (for example, if the transaction’s message is too long to fit into 2187 trytes).
- `add_inputs: (List[Address]) -> None`: Specifies inputs that can be used to fund transactions that spend IOTAs. The `ProposedBundle` will use these to create the necessary input transactions.
- You can use the `get_inputs` API command to find suitable inputs.

- `send_unspent_inputs_to: (Address) -> None`: Specifies the address that will receive unspent IOTAs. The `ProposedBundle` will use this to create the necessary change transaction, if necessary.
- `finalize: () -> None`: Prepares the bundle for PoW. Once this method is invoked, no new transactions may be added to the bundle.
- `sign_inputs: (KeyGenerator) -> None`: Generates the necessary cryptographic signatures to authorize spending the inputs. You do not need to invoke this method if the bundle does not contain any transactions that spend IOTAs.

Once the `ProposedBundle` has been finalized (and inputs signed, if necessary), invoke its `as_tryte_strings` method to generate the raw trytes that should be included in an `attach_to_tangle` API request.