Py-Authorize Documentation

Release 1.3.0.0

Vincent Catalano

Feb 26, 2018

Contents

Py-Authorize is a full-featured Python API for the Authorize.net payment gateway. Authorize.net offers great payment processing capabilities with a terribly incoherent API. Py-Authorize attempts to alleviate many of the problems programmers might experience with Authorize.net's'API by providing a cleaner, simpler and much more coherent API.

Py-Authorize supports most all of the Authorize.net's API functionality including:

- Advanced Integration Method (AIM)
- Customer Integration Manager (CIM)
- Transaction Detail API/Reporting
- Automated Recurring Billing API (ARB)

Here is a simple example of a basic credit card transaction.

```
import authorize

authorize.Configuration.configure(
    authorize.Environment.TEST,
    'api_login_id',
    'api_transaction_key',
)

result = authorize.Transaction.sale({
    'amount': 40.00,
    'credit_card': {
        'card_number': '4111111111111111',
        'expiration_date': '04/2014',
        'card_code': '343',
    }
})

result.transaction_response.trans_id
# e.g. '2194343352'
```

Py-Authorize is released under the MIT License.

Contents:

CHAPTER 1

Installation

1.1 Install with pip

If you are using pip, you can install the :app:'Py-Authorize' package using the following commands:

```
pip install py-authorize
```

1.2 Install with virtualenv

If you are using virtualenv to manage your packages, you can install :app:'Py-Authorize' using the following commands:

```
easy_install py-authorize
```

1.3 Install from source

Download or clone the source from Github and run setup.py install:

```
git clone http://github.com/vcatalano/py-authorize.git
cd py-authorize
python setup.py install
```

1.4 Requirements

Py-Authorize has only one external dependency:

- colander

If you want to build the docs or run the tests, there are additional dependencies, which are covered in the *Development* section.

CHAPTER 2

Getting Started

The first step when using the Py-Authorize API is to initialize the client with your Authorize.net API login name and transaction key. The initialization will only need to occur once in your application and must be setup before any other API calls are used.

2.1 Test Environment

```
import authorize

authorize.Configuration.configure(
    authorize.Environment.TEST,
    'api_login_id',
    'api_transaction_key',
)
```

In addition to the Authorize.net API login name and transaction key, the `configure` method also takes an `Environment` parameter. For development and testing configurations users should use the `Environment.TEST'` variable. For production configurations, users should use the `Environment.PRODUCTION` variable:

2.2 Production Environment

```
import authorize

authorize.Configuration.configure(
    authorize.Environment.PRODUCTION,
    'api_login_id',
    'api_transaction_key',
)
```

CHAPTER 3

Transactions

The primary purpose for any payment gateway is to provide functionality for taking payments for goods or services and charging a consumer. Py-Authorize's Transaction API provides all the functionality developers will need for all situations when developing a payment system.

3.1 Sale

The most common transaction type for credit cards is a ''sale''. During the transaction, the credit card is first authorized for the given transaction amount, if approved, it is automattically submitted for settlement.

Note: When performing a `sale` transaction, Py-Authorize is actually performing an `authCapture`.

3.1.1 Minimal Example

```
result = authorize.Transaction.sale({
    'amount': 40.00,
    'credit_card': {
        'card_number': '4111111111111111',
        'expiration_date': '04/2014',
    }
})

result.transaction_response.trans_id
# e.g. '2194343352'
```

Py-Authorize fully supports all Authorize.net gateway parameters for transactions.

3.1.2 Full Example

```
result = authorize.Transaction.sale({
    'amount': 56.00,
    'email': 'rob@robotronstudios.com',
    'credit_card': {
        'card_number': '4111111111111111',
        'card_code': '523',
        'expiration_date': '04/2014',
    },
    'shipping': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
    },
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
        'phone_number': '520-123-4567',
        'fax_number': '520-456-7890',
    },
    'tax': {
        'amount': 4.00,
        'name': 'Double Taxation Tax',
        'description': 'Another tax for paying double tax',
    },
    'duty': {
        'amount': 2.00,
        'name': 'The amount for duty',
        'description': 'I can''t believe you would pay for duty',
    },
    'line_items': [{
        'item_id': 'CIR0001',
        'name': 'Circuit Board',
        'description': 'A brand new robot component',
        'quantity': 5,
        'unit_price': 4.00,
        'taxable': 'true',
    }, {
        'item_id': 'CIR0002',
        'name': 'Circuit Board 2.0',
        'description': 'Another new robot component',
        'quantity': 1,
        'unit_price': 10.00,
        'taxable': 'true',
    }, {
        'item_id': 'SCRDRVR',
```

```
        'name': 'Screwdriver',
        'description': 'A basic screwdriver',
        'quantity': 1,
        'unit_price': 10.00,
        'taxable': 'true',
    }],
    'user_fields': [{
        'name': 'additionalDescription',
        'value': 'An additional description goes here...'
    }, {
        'name': 'moreInfo',
        'value': 'This is some more information...'
    }]
    'order': {
        'invoice_number': 'INV0001',
        'description': 'Just another invoice...',
    },
    'shipping_and_handling': {
        'amount': 10.00,
        'name': 'UPS 2-Day Shipping',
        'description': 'Handle with care',
    },
    'extra_options': {
        'customer_ip': '100.0.0.1',
    },
    'retail': {
        'market_type':0,
        'device_type':7,
    },
    'tax_exempt': False,
    'recurring': True,
})

result.transaction_response.trans_id
# e.g. '2194343353'
```

3.1.3 Card Present Example

If doing a card present transaction, track data can be passed in instead of a parsed credit card.

Note: It may still be useful to parse the track data in application logic to verify expiration date or card issuer.

```
result = authorize.Transaction.sale({
    'amount': 40.00,
    'track_data': {
        'track_1': '%B4111111111111111^OTERON/ROB^
↪1404101030052330000000000000000000000000000000?',
    }
})

result.transaction_response.trans_id
# e.g. '2194343352'
```

3.1.4 Minimal Bank Account Transaction

Transactions can also be ran against bank accounts.

> **Warning:** Since bank account (eCheck.net) transactions are handled differently from credit card transactions, you should avoid using the *auth* method when dealing with bank accounts. Only use the *sale* method when processing payments.

```
result = authorize.Transaction.sale({
    'amount': 40.00,
    'bank_account': {
        'routing_number': '322271627',
        'account_number': '00987467838473',
        'name_on_account': 'Rob Otron',
    },
})

result.transaction_response.trans_id
# e.g. '2194343357'
```

3.1.5 Full Transactions with Bank Accounts

```
result = authorize.Transaction.sale({
    'amount': 56.00,
    'email': 'rob@robotronstudios.com',
    'bank_account': {
        'customer_type': 'individual',
        'account_type': 'checking',
        'routing_number': '322271627',
        'account_number': '00987467838473',
        'name_on_account': 'Rob Otron',
        'bank_name': 'Evil Bank Co.',
        'echeck_type': 'WEB',
    },
    'shipping': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
    },
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
        'phone_number': '520-123-4567',
```

```
        'fax_number': '520-456-7890',
    },
    'tax': {
        'amount': 4.00,
        'name': 'Double Taxation Tax',
        'description': 'Another tax for paying double tax',
    },
    'duty': {
        'amount': 2.00,
        'name': 'The amount for duty',
        'description': 'I can''t believe you would pay for duty',
    },
    'line_items': [{
            'item_id': 'CIR0001',
            'name': 'Circuit Board',
            'description': 'A brand new robot component',
            'quantity': 5,
            'unit_price': 4.00,
            'taxable': 'true',
        }, {
            'item_id': 'CIR0002',
            'name': 'Circuit Board 2.0',
            'description': 'Another new robot component',
            'quantity': 1,
            'unit_price': 10.00,
            'taxable': 'true',
        }, {
            'item_id': 'SCRDRVR',
            'name': 'Screwdriver',
            'description': 'A basic screwdriver',
            'quantity': 1,
            'unit_price': 10.00,
            'taxable': 'true',
        }],
    'order': {
        'invoice_number': 'INV0001',
        'description': 'Just another invoice...',
    },
    'shipping_and_handling': {
        'amount': 10.00,
        'name': 'UPS 2-Day Shipping',
        'description': 'Handle with care',
    },
    'extra_options': {
        'customer_ip': '100.0.0.1',
    },
    'tax_exempt': False,
    'recurring': True,
})

result.transaction_response.trans_id
# e.g. '2194343358'
```

3.1.6 Transactions with CIM Data

Transactions can also be ran with stored customer payment profile information. When performing a transaction for a CIM managed payment profile, you must include the customer ID and payment profile ID. Additionally, you can include a customer's stored address ID as the shipping address for an order.

```
result = authorize.Transaction.sale({
    'amount': 56.00,
    'customer_id': '19086684',
    'payment_id': '17633614',
    'shipping_id': '14634122',
})

result.transaction_response.trans_id
# e.g. '2194343354'
```

3.1.7 Full Transactions Example with CIM Data

```
result = authorize.Transaction.sale({
    'amount': 56.00,
    'customer_id': '19086684',
    'payment_id': '17633614',
    'shipping_id': '14634122',
    'tax': {
        'amount': 4.00,
        'name': 'Double Taxation Tax',
        'description': 'Another tax for paying double tax',
    },
    'duty': {
        'amount': 2.00,
        'name': 'The amount for duty',
        'description': 'I can''t believe you would pay for duty',
    },
    'line_items': [{
            'item_id': 'CIR0001',
            'name': 'Circuit Board',
            'description': 'A brand new robot component',
            'quantity': 5,
            'unit_price': 4.00,
            'taxable': 'true',
        }, {
            'item_id': 'CIR0002',
            'name': 'Circuit Board 2.0',
            'description': 'Another new robot component',
            'quantity': 1,
            'unit_price': 10.00,
            'taxable': 'true',
        }, {
            'item_id': 'SCRDRVR',
            'name': 'Screwdriver',
            'description': 'A basic screwdriver',
            'quantity': 1,
            'unit_price': 10.00,
            'taxable': 'true',
        }],
    'order': {
```

```
        'invoice_number': 'INV0001',
        'description': 'Just another invoice...',
        'order_number': 'PONUM00001',
    },
    'shipping_and_handling': {
        'amount': 10.00,
        'name': 'UPS 2-Day Shipping',
        'description': 'Handle with care',
    },
    'extra_options': {
        'customer_ip': '100.0.0.1',
    },
    'tax_exempt': False,,
    'recurring': True,
})

result.transaction_response.trans_id
# e.g. '2194343355'
```

Note: The *email* field cannot be used in combination with the *customer_id* field. If the *customer_id* field is provided, the *email* field will be ignored during the transaction processing.

3.2 Auth

The `auth` method is equivalent to the the Authorize.net `authorizeOnly` method. When calling `auth`, the credit card is temporarily authorized for the given transaction amount without being submitted for settlement. This allows you to ensure you will be able to charge the card but hold off if in case you later no longer need to charge the card or need reduce the amount you plan to charge. In order to finalize the transaction charge, you must settle the transaction by using the `settle` transaction method.

This method takes the same parameters as the `sale` method.

3.2.1 Example

```
result = authorize.Transaction.auth({
    'amount': 40.00,
    'credit_card': {
        'card_number': '4111111111111111',
        'expiration_date': '04/2014,
    }
})

result.transaction_response.trans_id
# e.g. '2194343356'
```

The `auth` method takes the same values as as the `sale` method.

3.3 Settling

In order to finalize a previously authorized transaction, you must call the `settle` method with the transaction ID. When settling a transaction, the amount for the transaction can be changed as long as it is less than the original authorized amount.

3.3.1 Example

```
result = authorize.Transaction.settle('89798235')
```

The amount is not required if you want to settle the authorized amount. To settle a different amount, pass the amout as the second parameter.

```
result = authorize.Transaction.settle('89798235', 20.00)
```

3.4 Refund

This transaction type is used to refund a customer for a transaction that was originally processed and successfully settled through the payment gateway (it is the Authorize.net equivalent of a Credit).

When issuing a refund, Authorize.net requires the amount of the transaction, the last four digits of the credit card and the transaction ID. If you do not have the amount or last four digits of the credit card readily available, this information can be gotten using the `details` method.

3.4.1 Example

```
result = authorize.Transaction.refund({
    'amount': 40.00,
    'last_four': '1111',
    'transaction_id': '0123456789'
})
```

3.5 Void

This transaction type can be used to cancel either an original transaction that is not yet settled or an entire order composed of more than one transaction. A Void prevents the transaction or the order from being sent for settlement. You will only be able to void a transaction that is not already settled, expired, or failed.

3.5.1 Example

```
result = authorize.Transaction.void('0123456789')
```

3.6 Credit

Authorize.net provides the ability to issue refunds for transactions that were not originally submitted through the payment gateway (it is the Authorize.net equivalent of an Unlinked Credit). It also allows you to override restrictions set on basic credits, such as refunds for transactions beyond the 120-day refund limit.

Note: The ability to submit unlinked credits is not a standard payment gateway account feature. You must request the Expanded Credits Capability (ECC) feature by submitting an application to Authorize.net. More information on Unlinked Credits can be found under Authorize.net Transaction Types documentation.

3.6.1 Example

```
result = authorize.Transaction.credit({
    'amount': 120.00,
    'customer_id': '0987654321',
    'payment_id': '1348979152'
})
```

3.7 Details

This transaction type is used to get detailed information about one specific transaction based on the transaction ID.

3.7.1 Example

```
result = authorize.Transaction.details('0123456789')
```

3.8 List Transactions

This transaction type will return data for all transactions in a given batch.

3.8.1 Example

```
result = authorize.Transaction.list('0123456789')
```

Additionally, omitting the batch ID will return data for all transactions that are currently unsettled.

3.8.2 Example

```
result = authorize.Transaction.list()
```

CHAPTER 4

Customer

The `Customer` class provides an interface to Authorize.net's Customer Information Manager (CIM) API.

4.1 Create

When creating a customer profile, no information is actually needed. A random merchant ID is associated to the customer if none is provided. Once a user been created, address and payment information can then be associated to the profile.

4.1.1 Minimal Example

```
result = authorize.Customer.create()

result.customer_id
# e.g. '19086351'
```

4.1.2 Creating Profile with Basic Information

```
result = authorize.Customer.create({
    'email': 'rob@robotronstudios.com',
    'description': 'Rob the robot',
    'customer_type': 'individual',
})

result.customer_id
# e.g. '19086352'
```

4.1.3 Full Example

When creating a customer, additional shipping address and payment information can be provided as well.

```
result = authorize.Customer.create({
    'merchant_id': '8989762983402603',
    'email': 'rob@robotronstudios.com',
    'description': 'Rob the robot',
    'customer_type': 'individual',
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
        'phone_number': '520-123-4567',
        'fax_number': '520-456-7890',
    },
    'credit_card': {
        'card_number': '4111111111111111',
        'card_code': '456',
        'expiration_month': '04',
        'expiration_year': '2014',
    },
    'shipping': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
        'phone_number': '520-123-4567',
        'fax_number': '520-456-7890',
    }
})

result.customer_id
# e.g. '19086352'
```

4.1.4 Creating Profile from a Transaction

```
result = authorize.Transaction.auth({
    'amount': 40.00,
    'email': 'rob@robotronstudios.com',
    'credit_card': {
        'card_number': '4111111111111111',
        'expiration_date': '04/2014',
    }
})

trans_id = result.transaction_response.trans_id
```

```
# e.g. '2194343352'

result = authorize.Customer.from_transaction(trans_id)

result.profile.email
# rob@robotronstudios.com

result.customer_id
# e.g. '19086352'
```

4.2 Details

The `details` method returns the information for a given customer profile based on the customer ID.

The following information is returned in the result attribute dictionary:

- `profile.merchant_id`
- `profile.email`
- `profile.description`
- `profile.customer_type`
- `address_ids`
- `payment_ids`

```
result = authorize.Customer.details('19086352')
```

4.3 Update

Customer profile information can be easily updated on the server.

```
result = authorize.Customer.update('19086352', {
    'email': 'rob@robotronstudios.com',
    'description': 'Rob the robot',
    'customer_type': 'individual',
})
```

4.4 Delete

Deleting a customer will delete the customer profile along with all stored addresses and billing information.

```
result = authorize.Customer.delete('19086352')
```

4.5 List

The `list` method returns a list of all customer profile IDs.

```
result = authorize.Customer.list()

result.profile_ids
# e.g. ['16467005', '16467010', '16467092', '17556329']
```

CHAPTER 5

Credit Cards

Credit cards must be associated to a customer profile on the Authorize.net server. A credit card can be associated when a new customer is created, to see how this is handled refer to the *Customer API* documentation.

5.1 Create

To add a credit card to an existing user, the minimal amount of information required is the credit card number, the expiration date and the customer profile ID. The customer profile ID is passed as the first argument to the `create` method.

5.1.1 Minimal Example

```
result = authorize.CreditCard.create('customer_id', {
    'card_number': '4111111111111111',
    'expiration_date': '04/2014',
})

result.payment_id
# e.g. '17633318'
```

When creating a new credit card, the expiration month and date can be separate values.

```
result = authorize.CreditCard.create('customer_id', {
    'card_number': '4111111111111111',
    'expiration_month': '04',
    'expiration_year': '2014',
})

result.payment_id
# e.g. '17633319'
```

5.1.2 Full Example

When creating a new credit card, the billing address information can also be associated to the card.

```
result = authorize.CreditCard.create('customer_id', {
    'customer_type': 'individual',
    'card_number': '4111111111111111',
    'expiration_month': '04',
    'expiration_year': '2014',
    'card_code': '123',
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
        'phone_number': '520-123-4567',
        'fax_number': '520-456-7890',
    },
})

result.payment_id
# e.g. '17633319'
```

5.2 Details

The `details` method returns the information for a given customer payment profile. This method takes both the customer profile ID and customer payment profile ID.

The following information is returned in the result attribute dictionary:

- `payment_profile.payment_id`
- `payment_profile.customer_type`
- `payment_profile.payment.credit_card.card_number`
- `payment_profile.payment.credit_card.expiration_date`
- `payment_profile.bill_to.company`
- `payment_profile.bill_to.first_name`
- `payment_profile.bill_to.last_name`
- `payment_profile.bill_to.address`
- `payment_profile.bill_to.city`
- `payment_profile.bill_to.state`
- `payment_profile.bill_to.zip`
- `payment_profile.bill_to.country`
- `payment_profile.bill_to.phone_number`
- `payment_profile.bill_to.fax_number`

```
result = authorize.CreditCard.details('customer_id', '17633319')
```

5.3 Update

The `update` method will update the credit card information for a given payment profile ID. The method requires the customer profile ID, the payment profile ID and the new credit card information.

```
result = authorize.CreditCard.update('customer_id', '17633319', {
    'customer_type': 'individual',
    'card_number': '4111111111111111',
    'expiration_month': '04',
    'expiration_year': '2014',
    'card_code': '123',
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
        'phone_number': '520-123-4567',
        'fax_number': '520-456-7890',
    },
})
```

5.4 Delete

Deleting a customer credit card will remove the payment profile from the given customer.

```
result = authorize.CreditCard.delete('customer_id', '17633319')
```

5.5 Validate

Stored credit cards can be validated before attempting to run a transaction against them.

```
result = authorize.CreditCard.validate('customer_id', '17633319', {
    'card_code': '123',
    'validationMode': 'liveMode'
})
```

5.6 Transactions

For information on how to run transactions agains stored credit cards, please refer to the *Transaction* documentation.

CHAPTER 6

Bank Accounts

Bank accounts must be associated to a customer profile on the Authorize.net server. A bank account can be associated when a new customer is created, to see how this is handled refer to the *Customer API* documentation.

Note: The ability to process transactions from a bank account is not a standard gateway account feature. You must register for eCheck functionality seperately. For more information see Authorize.net's eCheck documentation.

6.1 Create

To add a bank account to an existing user, the minimal amount of information required is the routing number, account number, name on the account and the customer profile ID. The customer profile ID is passed as the first argument to the `create` method.

6.1.1 Minimal Example

```
result = authorize.BankAccount.create('customer_id', {
    'routing_number': '322271627',
    'account_number': '00987467838473',
    'name_on_account': 'Rob Otron',
})

result.payment_id
# e.g. '17633593'
```

6.1.2 Full Example

When creating a new bank account, the billing address information can also be associated to the account.

```
result = authorize.BankAccount.create('customer_id', {
    'customer_type': 'individual',
    'account_type': 'checking',
    'routing_number': '322271627',
    'account_number': '00987467838473',
    'name_on_account': 'Rob Otron',
    'bank_name': 'Evil Bank Co.',
    'echeck_type': 'CCD',
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
        'phone_number': '520-123-4567',
        'fax_number': '520-456-7890',
    },
})

result.payment_id
# e.g. '17633614'
```

6.2 Details

The `details` method returns the information for a given customer payment profile. This method takes both the customer profile ID and customer payment profile ID.

The following information is returned in the result attribute dictionary:

- `payment_profile.payment_id`
- `payment_profile.customer_type`
- `payment_profile.payment.bank_account.account_type`
- `payment_profile.payment.bank_account.routin_number`
- `payment_profile.payment.bank_account.account_number`
- `payment_profile.payment.bank_account.name_on_account`
- `payment_profile.payment.bank_account.bank_name`
- `payment_profile.payment.bank_account.echeck_type`
- `payment_profile.bill_to.company`
- `payment_profile.bill_to.first_name`
- `payment_profile.bill_to.last_name`
- `payment_profile.bill_to.address`
- `payment_profile.bill_to.city`
- `payment_profile.bill_to.state`
- `payment_profile.bill_to.zip`

- payment_profile.bill_to.country
- payment_profile.bill_to.phone_number
- payment_profile.bill_to.fax_number

```
result = authorize.BankAccount.details('customer_id', '17633614')
```

6.3 Update

The update method will update the bank account information for a given payment profile ID. The method requires the customer profile ID, the payment profile ID and the new bank account information.

```
result = authorize.BankAccount.update('customer_id', '17633614', {
    'customer_type': 'individual',
    'account_type': 'checking',
    'routing_number': '322271627',
    'account_number': '00987467838473',
    'name_on_account': 'Rob Otron',
    'bank_name': 'Evil Bank Co.',
    'echeck_type': 'CCD',
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
        'phone_number': '520-123-4567',
        'fax_number': '520-456-7890',
    },
})
```

6.4 Delete

Deleting a customer bank account will remove the payment profile from the given customer.

```
result = authorize.BankAccount.delete('customer_id', '17633319')
```

6.5 Transactions

For information on how to run transactions against stored credit cards, please refer to the *Transaction* documentation.

CHAPTER 7

PayPal Express Checkout

Authorize.net now provides functionality for PayPal Express Checkout. With PayPal Express Checkout, you can accept payments with PayPal while utilizing Authorize.net's reporting functionality.

For more detailed information about how the PayPal Express Checkout process works with Authorize.net, visit the official *PayPal Express Checkout* documentation.

7.1 Additional API Flow Functions

In order to handle the additional steps required by the PayPal Express Checkout flow process, two additional functions have been added to the Transaction API: `Transaction.auth_continue` and `Transaction.sale_continue`. These functions refer to `Authorize Only, Continue` and `Authorize and Capture, Continue` requests, respectively.

7.2 Transaction Flow Sequence Example 1

1. Authorization Only
2. Get Details (recommended for shipping)
3. Authorization Only, Continue
4. Prior Authorization Capture
5. Refund (optional)

```
result = authorize.Transaction.auth({
    'amount': 40.00,
    'pay_pal': {
        'success_url': 'https://my.server.com/success.html',
        'cancel_url': 'https://my.server.com/cancel.html',
        'locale': 'US',
        'header_image': 'https://usa.visa.com/img/home/logo_visa.gif',
```

```
        'flow_color': 'FF0000'
    },
})

result.transaction_response.trans_id
# e.g. 'transaction_id'

result.secure_acceptance.secure_acceptance_url
# e.g. https://www.paypal.com/cgibin/webscr?cmd=_express-checkout&token=EC-
↪4WL17777V4111184H

# (optional) get shipping information for order
details = authorize.Transaction.details('transaction_id')

authorize.Transaction.auth_continue('transaction_id', 'payer_id')

authorize.Transaction.settle('transaction_id')

# (optional) refund the transaction
authorize.Transaction.refund('transaction_id')
```

7.3 Transaction Flow Sequence Example 2

1. Authorization Only
2. Get Details (recommended for shipping)
3. Authorization Only, Continue
4. Void

```
result = authorize.Transaction.auth({
    'amount': 40.00,
    'pay_pal': {
        'success_url': 'https://my.server.com/success.html',
        'cancel_url': 'https://my.server.com/cancel.html',
        'locale': 'US',
        'header_image': 'https://usa.visa.com/img/home/logo_visa.gif',
        'flow_color': 'FF0000'
    },
})

result.transaction_response.trans_id
# e.g. 'transaction_id'

result.secure_acceptance.secure_acceptance_url
# e.g. https://www.paypal.com/cgibin/webscr?cmd=_express-checkout&token=EC-
↪4WL17777V4111184H

# (optional) get shipping information for order
details = authorize.Transaction.details('transaction_id')

authorize.Transaction.auth_continue('transaction_id', 'payer_id')

authorize.Transaction.void('transaction_id')
```

7.4 Transaction Flow Sequence Example 3

1. Authorization and Capture
2. Get Details (recommended for shipping)
3. Authorization and Capture, Continue
4. Refund (optional)

```
result = authorize.Transaction.sale({
    'amount': 40.00,
    'pay_pal': {
        'success_url': 'https://my.server.com/success.html',
        'cancel_url': 'https://my.server.com/cancel.html',
        'locale': 'US',
        'header_image': 'https://usa.visa.com/img/home/logo_visa.gif',
        'flow_color': 'FF0000'
    },
})

result.transaction_response.trans_id
# e.g. 'transaction_id'

result.secure_acceptance.secure_acceptance_url
# e.g. https://www.paypal.com/cgibin/webscr?cmd=_express-checkout&token=EC-
↪4WL17777V4111184H

# (optional) get shipping information for order
details = authorize.Transaction.details('transaction_id')

authorize.Transaction.sale_continue('transaction_id', 'payer_id')

authorize.Transaction.refund('transaction_id')
```

CHAPTER 8

Address

The Address API manages customer shipping addresses for Authorize.net's Customer Information Manager (CIM). Addresses must be associated to a customer profile on the Authorize.net server. An address can be associated when a new customer is created, to see how this is handled refer to the *Customer API* documentation.

8.1 Create

To associate an address to an existing customer use the *create* method. When creating an address, you must provide the customer profile ID as the first argument. No fields are required when creating an address, however, at least one field must be provided.

```
result = authorize.Address.create('customer_id', {
    'first_name': 'Rob',
    'last_name': 'Oteron',
    'company': 'Robotron Studios',
    'address': '101 Computer Street',
    'city': 'Tucson',
    'state': 'AZ',
    'zip': '85704',
    'country': 'US',
    'phone_number': '520-123-4567',
    'fax_number': '520-456-7890',
})

result.address_id
# e.g. '17769620'
```

8.2 Details

The *details* method returns the information for a given customer address. You must provide the both customer profile ID and the customer address ID respectively.

The following information is returnd in the result attribute dictionary:

- address.first_name
- address.last_name
- address.company
- address.address
- address.city
- address.state
- address.zip
- address.country
- address.phone_number
- address.fax_number

```
result = authorize.Address.details('customer_id', '17769620')

result.address_id
# e.g. '17769620'
```

8.3 Update

The update method will update the address information for a given address ID. The method requires the customer profile ID, the customer address ID and the updated customer address information.

```
result = authorize.Address.create('customer_id', '17769620', {
    'first_name': 'Rob',
    'last_name': 'Oteron',
    'company': 'Robotron Studios',
    'address': '101 Computer Street',
    'city': 'Tucson',
    'state': 'AZ',
    'zip': '85704',
    'country': 'US',
    'phone_number': '520-123-4567',
    'fax_number': '520-456-7890',
})
```

8.4 Delete

Deleting a customer address will remove the address information associated the customer.

```
authorize.Address.delete('customer_id', '17769620')
```

CHAPTER 9

Recurring

The Py-Authorize Recurring API is used to integrate with Authorize.net's Automated Recurring Billing (ARB) subscription-based payment service. It provides all functionality for managing recurring billing against credit cards and bank accounts.

9.1 Create

Authorize.net's ARB service functions seperately from the Customer Information Management API. This means you cannot create recurring payments for stored customers, credit cards or bank accounts. Instead, you will need to provide all customer and payment information explicitly.

9.1.1 Minimal Example

```
result = authorize.Recurring.create({
    'amount': 45.00,
    'interval_length': 1,
    'interval_unit': 'months',
    'credit_card': {
        'card_number': '4111111111111111',
        'expiration_date': '04-2014',
        'card_code': '456',
    },
})

# result.subscription_id
# e.g. '1725604'
```

In this example, the customer will be charged $45.00 every month until the subscription is canceled or the payment gateway can no longer process the payment method (e.g. the card has expired). Authorize.net only permits interval units of *days* or *years*.

To specify a limited number of occurrences, use the *total_occurrences* parameter.

```
result = authorize.Recurring.create({
    'amount': 45.00,
    'interval_length': 14,
    'interval_unit': 'days',
    'total_occurrences': 52,
    'credit_card': {
        'card_number': '4111111111111111',
        'expiration_date': '04-2014',
        'card_code': '456',
    },
})

# result.subscription_id
# e.g. '1725605'
```

9.1.2 Full Example

Recurring payments can also be configured with customer bank accounts. The following example shows all recurring payment parameters available.

```
result = authorize.Recurring.create({
    'amount': 45.00,
    'name': 'Ultimate Robot Supreme Plan',
    'total_occurrences': 30,
    'interval_length': 2,
    'interval_unit': 'months',
    'trial_amount': 30.00,
    'trial_occurrences': 2,
    'bank_account': {
        'customer_type': 'individual',
        'account_type': 'checking',
        'routing_number': '322271627',
        'account_number': '00987467838473',
        'name_on_account': 'Rob Otron',
        'bank_name': 'Evil Bank Co.',
    },
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
    },
    'shipping': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
    },
```

```
    'order': {
        'invoice_number': 'INV0001',
        'description': 'Just another invoice...',
    },
    'customer': {
        'merchant_id': '1234567890',
        'email': 'rob@robotronstudios.com',
        'description': 'I am a robot',
    },
})

# result.subscription_id
# e.g. '1725628'
```

9.2 Details

To the get the details of a recurring payment, use the *details* method.

```
result = authorize.Recurring.details('1725628')

# result.subscription.profile.customer_id
# e.g. '1806948040'

# result.status
# e.g. 'active'
```

9.3 Status

To the get the status of a recurring payment, use the *status* method.

```
result = authorize.Recurring.status('1725628')

# result.status
# e.g. 'active'
```

9.4 Update

The *update* method takes the same parameters as the *create* method. However, once recurring payments have started, there are certain exceptions.

- The subscription *start_date* may only be updated if no successful payments have been completed.
- The subscription *interval_length* and *interval_unit* may not be updated. Instead, you must create a new subscription if you want different values for these parameters.
- The number of *trial_occurrences* may only be updated if the subscription has not yet begun or is still in the trial period.
- If the *start_date* is the 31st, and the interval is monthly, the billing date is the last day of each month (even when the month does not have 31 days).

When updating a recurring payment, you must pass in the subscription ID of the payment you wish to update along with the new subscription information.

```
result = authorize.Recurring.update('1725628', {
    'name': 'Ultimate Robot Supreme Plan',
    'amount': 45.00,
    'total_occurrences': 30,
    'trial_amount': 30.00,
    'trial_occurrences': 2,
    'credit_card': {
        'card_number': '4111111111111111',
        'expiration_date': '04-2014',
        'card_code': '456',
    },
    'billing': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
    },
    'shipping': {
        'first_name': 'Rob',
        'last_name': 'Oteron',
        'company': 'Robotron Studios',
        'address': '101 Computer Street',
        'city': 'Tucson',
        'state': 'AZ',
        'zip': '85704',
        'country': 'US',
    },
    'order': {
        'invoice_number': 'INV0001',
        'description': 'Just another invoice...',
    },
    'customer': {
        'merchant_id': '1234567890',
        'email': 'rob@robotronstudios.com',
        'description': 'I am a robot',
    },
})
```

9.5 Cancel

To cancel a recurring payment, pass the subscription ID to the *cancel* method.

```
authorize.Recurring.delete('1725628')
```

CHAPTER 10

Batch

Transactions are batched and sent for settlement on a daily basis. Py-Authorize provides basic batch methods based on Authorize.net's reporting API.

10.1 List

The *list* method returns the batch ID, Settlement Time and other batch statistics for all settled batches within a range of dates.

```
result = authorize.Batch.list({
    'start': '2012-01-01',
    'end': '2012-05-31',
})
```

If the *start* and *end* dates are not specified, the *list* method will return the batches processed in the past 24 hours.

```
result = authorize.Batch.list()
```

10.2 Details

The *details* method returns batch statistics for a given batch ID.

```
result = authorize.Batch.details('2552096')
```

CHAPTER 11

Advanced

Out-of-the-box Py-Authorize provides some very basic and powerful functionality. For some users and applications, more advanced API functionality may be needed. This sections provides an overview and documentation for some of those features.

11.1 APIs

When configuring Py-Authorize with the *Configuration* global variable, you are actually instantiating a single instance of the *authorize_api* class. *authorize.Address*, *authorize.BankAccount*, *authorize.CreditCard*, *authorize.Customer* and *authorize.Recurring* are all wrappers for accessing this globally configured API. You can access the API explicitly through the *Configuration.api* class member. For example, to perform a basic sale transaction with a credit card using the API you would use the following method:

```
result = authorize.Configuration.api.transaction.sale({
    'amount': 40.00,
    'credit_card': {
        'card_number': '4111111111111111',
        'expiration_date': '04/2014',
    }
})
```

Each *authorize_api* instance contains the following members for performing API calls:

- `api.customer`
- `api.credit_card`
- `api.bank_account`
- `api.address`
- `api.recurring`
- `api.batch`
- `api.transaction`

11.2 Multiple Gateway Configurations

For some payment applications, there may be a need to support multiple gateways. With Py-Authorize, you can instantiate any number of payment gateway configurations.

11.2.1 Example

```
configuration_1 = authorize.Configuration(
    Environment.PRODUCTION,
    'api_login_id',
    'api_transaction_key',
)

configuration_2 = authorize.Configuration(
    Environment.PRODUCTION,
    'another_api_login_key',
    'another_api_transaction_key',
)
```

Once a new configuration has been created, you can make use of each configuration object's *api* members as outlined in *APIs*

CHAPTER 12

Development

Any development help is greatly appreciated. If you have have any new features, bug fixes or documentation improvements please feel free to contribute.

12.1 Getting started

To start developing on this project, fork this project on our Github page and install from source using the instructions in *Install*. Additionally, you will need to install the following dependencies for running test and compiling documentation.

- nose
- tox (for testing multiple versions of Python)
- sphinx (for documentation)
- sphinx_rtd_theme (documentation theme)

12.2 Running Tests

This project has been configured to use the Nose testing framework and Tox for automation. The following command will run all tests for the project. Since many of the tests connect to the Authorize.net server, running the tests may take quite a few seconds.

```
nosetests
```

To run only local tests, you can use the following command:

```
nosetests -a '!live_tests'
```

To local tests for Python versions 2.7, 3.3, 3.4, 3.5 and PyPy:

```
tox
```

12.3 Authorize.net documentation

The Authorize.net documentation can be overly verbose and very inconsistent with the implementations of many of its features. You can view the documentation by visiting the following links:

- Developer site
- Advanced Integration Method
- Customer Information Manager
- Automated Recurring Billing

12.4 Submitting bugs and patches

All bug reports, new feature requests and pull requests are handled through this project's Github issues page.