kount Documentation

Release 1.0.0

Kount Inc.

Apr 08, 2022

Contents

CHAPTER 1

Welcome to the kount-ris-python-sdk!

1.1 Kount Python RIS SDK

Contains the Kount Python SDK, tests, and build/package routines.

1. What is this repository for?

   Contains sources, tests, and resources for the Kount Python SDK SDK version: 1.0.0 Python 2.7.13 and 3.5.3, 3.6.1

2. How do I get set up?

```
pip install kount_ris_sdk
```

3. How to run integration tests against the installed sdk library.

   First, you need to obtain configuration key from Kount.

   Download the source code from https://github.com/Kount/kount-ris-python-sdk

   Go to the root directory of the project and execute:

```
pip install .[test]

pytest tests --conf-key={KEY}
# or set shell environment
export CONF_KEY={KEY}
pytest tests
```

Running specific test:

```
pytest tests/test_file.py
```

For more verbosity and code coverage run the following:

```
pytest -s -v --cov=kount tests/
```

4. Setting up IDE projects

- Komodo IDE/Edit, Scite, Visual Studio - have automatic python integration

5. Who do I talk to?
    1. Repo owner or admin
    2. Other community or team contact

CHAPTER 2

SDK Features

The Kount RIS Python SDK provides means for:

- building and sending requests to the RIS service
- client-side data verification
- sensitive information protection.

CHAPTER 3

Building requests

The SDK contains several object model classes representing different request types as well as many enumeration-like objects which can be used as request parameter values.

The `Client` class accomplishes client-server-client communication, secured TLS v1.2 for Python 2.7.13 and 3.6.x.

Older Python versions that do not implement TLSv1.2 will be prohibited from accessing PyPI.

See below for instructions to check your interpreter's TLS version. To check your Python interpreter's TLS version:

```
python -c "import requests; print(requests.get('https://www.howsmyssl.com/a/check',␣
↪verify=False).json()['tls_version'])"
```

If you see **TLS 1.2**, your interpreter's TLS is up to date. If you see "TLS 1.0" or an error like "tlsv1 alert protocol version", then you must upgrade. Mac users should pay special attention. So far, the system Python shipped with MacOS does not yet support TLSv1.2 in any MacOS version. Fortunately, it's easy to install a modern Python alongside the MacOS system Python. Either download Python 3.6 from python.org, or for Python 2.7 with the latest TLS, use Homebrew. Both methods of installing Python will continue working after June 2018.

The reason Python's TLS implementation is falling behind on macOS is that Python continues to use OpenSSL, which Apple has stopped updating on macOS. In the coming year, the Python Packaging Authority team will investigate porting pip to Apple's own "SecureTransport" library as an alternative to OpenSSL, which would allow old Python interpreters to use modern TLS with pip only.

CHAPTER 4

Client-side data verification

The Kount RIS Python SDK provides means for validation of most of the request parameters, as well as ensuring presence of required parameters for each request type.

CHAPTER 5

Sensitive information protection

The SDK utilizes specific hashing methods to encrypt and transmit sensitive client data like credit card and gift card numbers.

We provide a couple of ways to obtain the Python SDK: either pip, ordownload the package from github.

CHAPTER 6

Pip

```
pip install kount_ris_sdk
```

CHAPTER 7

Download the SDK from GitHub

Kount allows developers to download the SDK library from the Python SDK github repository. Please follow the next steps if you need help with this process:

- Clone the SDK repository to your machine
- use your preferred Git client
- console:
  - `git clone https://github.com/Kount/kount-ris-python-sdk.git`
  - `git clone git@github.com:Kount/kount-ris-python-sdk.git`
- Use your favorite python IDE or - Komodo IDE/Edit - Scite - Visual Studio

CHAPTER 8

Making requests to the Kount RIS

CHAPTER 9

Configuration prerequisites and requirements

Before you make your RIS call, you need to have received (or created) the following data from Kount:

- Merchant ID, 6-digit integer, referenced as `MERCHANT_ID` in code snippets
- Site ID, *url*
- URL for (test) RIS calls - `url_api`, currently `url_api = "https://risk.beta.kount.net"` in test_api_kount.py
- Hashing configurationKey used to encrypt sensitive data, configurationKey - must be configured with `kount.config.SDKConfig.setup()` call:
- API key, a JWT key used for authentication, *key* parameter in class Client client.py, used to perform communication with the RIS server.

CHAPTER 10

Creating request objects

There are two types of requests that can be performed through the SDK - `Inquiry` and `Update`. Those have various modes which will be discussed later on.

The usual structure of a `Request` usually consists of three parts:

- Information about **the merchant and processing instructions** for the RIS service
- Information about **the customer making a purchase**: personal data, geo-location, etc.
- Information about **the purchase**: product name, category, quantity, price

Let's create a sample `Inquiry` object and then send it to the RIS service. **/see test_inquiry.py/**

```
#python
import uuid
import logging
from kount.client import Client
from kount.config import SDKConfig
from kount.inquiry import Inquiry
from kount.request import InquiryMode, MerchantAcknowledgment, CurrencyType
from kount.util.address import Address
from kount.util.cartitem import CartItem
from kount.util.payment import CardPayment

MERCHANT_ID = 999667
EMAIL_CLIENT = "customer.name@email.com"
SHIPPING_ADDRESS = Address("567 West S2A1 Court North", "",
                           "Gnome", "AK", "99762", "US")
PTOK = "0007380568572514"
SITE_ID = "192.168.32.16"
URL_API = "https://kount.ris/url"
API_KEY = "real api key"
PROVIDED_CONFIGURATION_KEY = b'replace-this-with-real-one'

SDKConfig.setup(PROVIDED_CONFIGURATION_KEY)
```

(continues on next page)

(continued from previous page)

```
def evaluate_inquiry():
    session_id = generate_unique_id()[:32]
    inquiry = Inquiry()

    # set merchant information, see default_inquiry() in test_basic_connectivity.py
    inquiry.set_merchant(MERCHANT_ID)
    inquiry.set_request_mode(InquiryMode.DEFAULT)
    inquiry.set_merchant_acknowledgment(MerchantAcknowledgment.TRUE)
    inquiry.set_website("DEFAULT")

    # set customer information
    inquiry.set_unique_customer_id(session_id[:20])
    inquiry.set_ip_address(SITE_ID)
    payment = CardPayment(PTOK, khashed=False)    # credit-card-number
    # or for khashed token:
    # payment = CardPayment(PTOK)    # credit-card-number, khashed=True *default value*
    inquiry.set_payment(payment)
    inquiry.set_customer_name("SdkTestFirstName SdkTestLastName")
    inquiry.set_email_client(EMAIL_CLIENT)
    inquiry.set_shopping_cart(SHIPPING_ADDRESS)

    # set purchase information
    inquiry.set_currency(CurrencyType.USD)
    inquiry.set_total('123456')
    cart_items = list()
    cart_items.append(CartItem("SPORTING_GOODS", "SG999999",
                               "3000 CANDLEPOWER PLASMA FLASHLIGHT",
                               '2', '68990'))
    inquiry.set_shopping_cart(cart_items)

    client = Client(URL_API, API_KEY)
    response = client.process(inquiry)

    # do stuff with response


# method use for creating unique session id

def generate_unique_id():
    return str(uuid.uuid4()).replace('-','').upper()


def setup_logging():

    req = logging.getLogger('kount.request')
    req.setLevel(logging.DEBUG)
    reqh = logging.FileHandler('request.log')
    reqh.setLevel(logging.DEBUG)
    req.addHandler(reqh)


    cli = logging.getLogger('kount.client')
    cli.setLevel(logging.DEBUG)
    clih = logging.FileHandler('client.log')
    clih.setLevel(logging.DEBUG)
    cli.addHandler(clih)
```

(continues on next page)

(continued from previous page)

```
    resp = logging.getLogger('kount.response')
    resp.setLevel(logging.DEBUG)
    resph = logging.FileHandler('response.log')
    resph.setLevel(logging.DEBUG)
    resp.addHandler(resph)


# calling the evaluate_inquiry method
evaluate_inquiry()
```

CHAPTER 11

Explanation of the request

Here is a short description of what's going on during request creation, following the numbered comments in code

1. Creating the communication client, requires the RIS service url and provided API key. The API key is set as request header for the network request.
2. Setting the request mode. As mentioned previously, there are several request modes and **InquiryMode.INITIAL_INQUIRY** is the most used one. Please check the *Advanced* page for more information on request modes.
3. Setting a session identifier. This ID should be unique for a 30-day span and is used to track all changes regarding the purchase described in the request. More information on the *Advanced* page.
4. IP address of the customer. The merchant can discover it or it can be obtained through the *Data Collector* service.
5. Set this to a correct credit number or select another payment method (for test purposes).
6. The total purchase amount represented in the lowest possible currency denomination (*example: cents for US Dollars*)
7. Different payment types /user defined/ can be created with **NewPayment** or **Payment**:

```
NewPayment(payment_type="PM42", payment_token=token, khashed=True)
Payment("PM42", token, False)
Payment("PM42", token, True)
```

Good examples - test_bed_examples.py

CHAPTER 12

Response description

After a merchant has posted RIS information to Kount, a key-value pair string will be returned back to the merchant. The RIS response format will be the same that was specified in the RIS request, with the default being named pairs. Each data field must be invoked by getter methods on the `Response` object from the SDK. The merchant can then use the RIS response to automate the order management process by keying off of the `AUTO` field and can utilize any of the additional data returned for internal processing.

An important use of the RIS response is the ability to view any warnings or errors that were made during the RIS post from the merchant. All warnings will be displayed in the response and if errors do occur the RIS response will be returned with a `MODE = E` /if inquiry.params[“FRMT”] is not set/ or {“MODE”: “E”, “ERRO”: “201”} /if inquiry.params[“FRMT”] = “JSON”/. More information on warnings and errors can be found at the *Troubleshooting* section.

Response.get_errors() returns error list.

CHAPTER 13

SDK Integration & Unit Tests

CHAPTER 14

Integration tests

1. test_ris_test_suite.py
2. test_basic_connectivity.py
3. test_api_kount.py
4. json_test.py - Example curl call:

```
curl -k -H "X-Kount-Api-Key: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJpc3MiOiI5OTk2NjYiLCJhdWQiOiJLb3VudC4xIiwiaWF0IjoxNDk0NTM0Nzk5LCJzY3AiOnsia2EiOm51bGwsImtjIjpudW
↪eMmumYFpIF-d1up_mfxA5_VXBI41NSrNVe9CyhBUGck" -d "MODE=Q&LAST4=2514&PROD_
↪ITEM[]=SG999999&PROD_DESC[]=3000+CANDLEPOWER+PLASMA+FLASHLIGHT&
↪S2NM=SdkTestShipToFirst+SdkShipToLast&PTOK=0007380568572514&AUTH=A&IPAD=4.127.51.
↪215&B2CI=Albuquerque&S2CC=US&SESS=088E9F4961354D4F90041988B8D5C66B&TOTL=123456&PROD_
↪QUANT[]=2&B2CC=US&S2CI=Gnome&AVST=M&EMAL=curly.riscaller15%40kountqa.com&AVSZ=M&
↪S2PC=99762&S2EM=sdkTestShipTo%40kountsdktestdomain.com&S2ST=AK&FRMT=JSON&VERS=0695&
↪B2PC=87101&ORDR=088E9F496135&PROD_TYPE[]=SPORTING%5FGOODS&B2PN=555+867-5309&
↪S2PN=208+777-1212&NAME=Goofy+Grumpus&MACK=Y&SITE=DEFAULT&PROD_PRICE[]=68990&
↪UAGT=Mozilla%2F5.0+%28Macintosh%3B+Intel+Mac+OS+X+10%5F9%5F5%29+AppleWebKit%2F537.
↪36+%28KHTML%2C+like+Gecko%29+Chrome%2F37.0.2062.124+Safari%2F537.36&CVVR=M&
↪CASH=4444&B2ST=NM&ANID=&MERC=999666&CURR=USD&S2A1=567+West+S2A1+Court+North&
↪B2A1=1234+North+B2A1+Tree+Lane+South&PTYP=CARD&UNIQ=088E9F4961354D4F9004" https://
↪risk.beta.kount.net
```

should result in this response:

```
{"VERS":"0695","MODE":"Q","TRAN":"PTPN0Z04P8Y6","MERC":"999666","SESS":
↪"088E9F4961354D4F90041988B8D5C66B",
"ORDR":"088E9F496135","AUTO":"R","SCOR":"29","GEOX":"US","BRND":null,"REGN":null,"NETW
↪":"N","KAPT":"N","CARDS":"1",
"DEVICES":"1","EMAILS":"1","VELO":"0","VMAX":"0","SITE":"DEFAULT","DEVICE_LAYERS":"...
↪.","FINGERPRINT":null,
"TIMEZONE":null,"LOCALTIME":" ","REGION":null,"COUNTRY":null,"PROXY":null,"JAVASCRIPT
↪":null,"FLASH":null,"COOKIES":null,
"HTTP_COUNTRY":null,"LANGUAGE":null,"MOBILE_DEVICE":null,"MOBILE_TYPE":null,"MOBILE_
↪FORWARDER":null,
```

(continues on next page)

(continued from previous page)


```
"VOICE_DEVICE":null,"PC_REMOTE":null,"RULES_TRIGGERED":1,"RULE_ID_0":"1024842","RULE_
↪DESCRIPTION_0":
"Review if order total > $1000 USD","COUNTERS_TRIGGERED":0,"REASON_CODE":null,"DDFS
↪":null,"DSR":null,"UAS":null,
"BROWSER":null,"OS":null,"PIP_IPAD":null,"PIP_LAT":null,"PIP_LON":null,"PIP_COUNTRY
↪":null,"PIP_REGION":null,"PIP_CITY":null,
"PIP_ORG":null,"IP_IPAD":null,"IP_LAT":null,"IP_LON":null,"IP_COUNTRY":null,"IP_REGION
↪":null,"IP_CITY":null,"IP_ORG":null,"WARNING_COUNT":0}
```


The Kount RIS Python SDK comes with a suite of integration tests, covering the various request modes and service behavior. Some of the internal SDK functionality is also covered by those test cases.

Each Kount client, upon negotiating an NDA, is going to receive the following configuration data:

- RIS server URL
- Merchant ID
- API key
- configurationKey used in encrypting sensitive data.

14.1 Setting configurationKey

In order to run the set of integration tests, it is required to provide the configurationKey either with command line argument (–conf-key) or with exporting environment variable CONF_KEY

Within the test suite test_basic_connectivity.py, there's a test class named TestBasicConnectivity. Inside, there are two test cases, expecting predefined results (check the *Predictive Response* section). There are other options that have default values, although they can be overwritten for the test run:

- –merchant_id
- –api-key
- –api-url

14.2 unit tests:

1. test_address.py
2. test_payment.py
3. test_inquiry.py
4. test_ris_validation_exception.py
5. test_ris_validator.py
6. test_validation_error.py
7. test_xmlparser.py
8. test_khash.py

14.3 resources:

1. resources/validate.xml - Kount xml, used for request's validation
2. resources/correct_key_cryp.py- sha-256 of the correct configurationKey, used for validation

CHAPTER 15

Connection-troubleshooting

The team has developed a bunch of tools for connection diagnostics - host visibility, latency check, establishing connection, TLS/SSL-related information.

Those tools can be found on a separate Github repo - kount-ris-debug-utils

CHAPTER 16

Advanced

In this section we will discuss SDK peculiarities in greater detail. Those include different request types, session-related parameters, as well as descriptions of the mandatory and most of the optional parameters.

A couple of SDK-related tools also are worth mentioning. Those are instruments for client data collection, network and request troubleshooting.

- *Request types* - all request types supported by the Kount RIS SDK and their designation
- *Session related Parameters* - parameters used to track the changes for a given purchase
- *Payment Types*
- *Data Collector*
- *Troubleshooting*

CHAPTER 17

Request types and Parameters

Here we discuss the request types supported by the Kount RIS Python SDK and mandatory/optional parameters.

CHAPTER 18

Request types

There are two major `Request` types: `Inquiry` and `Update`. Their handling by the RIS service can be configured by setting the `MODE` parameter to the correct value.

`Inquiry` type should be used for initial registration of the purchase in the Kount system. It has four available values for the `MODE` parameter.

`Inquiry` MODE	SDK Constant	Description
`Q`	`InquiryMode.DEFAULT`	Default inquiry mode, internet order type
`P`	`InquiryMode.PHONE`	Used to analyze a phone-received order
`W`	`InquiryMode.WITH_THRE SHOLDS`	Kount Central full inquiry with returned thresholds
`J`	`InquiryMode.JUST_THRE SHOLDS`	Kount Central fast inquiry with just thresholds

`Update` type should be used whenever there are changes to a given order and the merchant wants them reflected into the Kount system. `Update` has two available values for the `MODE` parameter.

Update	SDK Constant	Description
`U`	`UpdateMode.NO_RE SPONSE`	Default update mode, only sends the update event
`X`	`UpdateMode.WITH_ RESPONSE`	Sends the update event and RIS service returns a status response

CHAPTER 19

Mandatory parameters

Parameter name	Setter	Q	P	W	J	U	X
MODE	set_mode	Y	Y	Y	Y	Y	Y
VERS	version_set	Y	Y	Y	Y	Y	Y
MERC	merchant_set	Y	Y	Y	Y	Y	Y
SITE	website	Y	Y	Y			
SESS	session_set	Y	Y	Y		Y	Y
CURR	currency_set	Y	Y	Y	Y		
TOTL	total_set	Y	Y	Y	Y		
MACK	merchant_acknowledgment_set	Y	Y	Y			
CUSTOMER_ID	kount_central_customer_id			Y	Y		
PTYP		Y	Y	Y	Y		
IPAD	ip_address | Y		Y	Y	Y		
TRAN	set_transaction_id					Y	Y
PROD_TYPE	**warning**	Y	Y	Y			
PROD_ITEM	**warning**	Y	Y	Y			
PROD_QUANT	**warning**	Y	Y	Y			
PROD_PRICE	**warning**	Y	Y	Y			
ANID	**warning**		Y				

warning Parameters marked with this warning sign are the shopping cart parameters. They are bulk-set by the `Inquiry.set_cart(cart)` method. If shopping cart contains more than one entry, values for each parameter are transformed to single concatenated strings and then set to the corresponding parameter.

CHAPTER 20

Optional parameters

The Kount RIS Python SDK provides a huge number of optional request parameters to increase precision when making a decision about a given purchase / order.

Only the most interesting parameters will be mentioned here. A comprehensive listing of all SDK-set parameters can be found on the python-doc pages for `Request`, `Inquiry`, and `Update` classes.

- `AUTH`: Authorization Status returned to merchant from processor. Acceptable values for the `AUTH` field are `A` for Authorized or `D` for Decline. In orders where `AUTH = A` will aggregate towards order velocity of the persona while orders where `AUTH = D` will decrement the velocity of the persona.
- `AVST`: Address Verification System Street verification response returned to merchant from processor. Acceptable values are `M` for match, `N` for no-match, or `X` for unsupported or unavailable.
- `AVSZ`: Address Verification System Zip Code verification response returned to merchant from processor. Acceptable values are `M` for match, `N` for no match, or `X` for unsupported or unavailable.
- `CVVR`: Card Verification Value response returned to merchant from processor. Acceptable values are `M` for match, `N` for no-match, or `X` unsupported or unavailable.
- `LAST4`: Last 4 numbers of Credit Card Value.

CHAPTER 21

User-defined fields

Kount provides a way for merchants to include additional information related to their business that may not be a standard field in Kount by creating user defined fields. UDFs are created in the Agent Web Console by browsing to the Fraud Control tab and clicking on User Defined Fields. Once you have defined the UDF in the AWC you will be able to pass this data into Kount via an array called UDF as key-value pairs where the label is the key and the data passed in is the value. The maximum number of UDFs that can be created is 500, and response time for evaluating transactions will degrade as more UDFs are added. There are four data types available for user defined fields.

Attribute	Size	Description	Example
`UDF[NUMERIC_LABE L] = value`	1-255	Numbers, negative signs, and decimal points	UDF[FREQUENCY] = 107.9
`UDF[ALPHA_NUMERIC_LABEL = value`	1-255	Letters, numbers, or both	UDF[COUPON] = BUY11
`UDF[DATE_LABEL] = value`	1-20	Formatted as `YYYY-MM-DD` or `YYYY-MM-DD HH:MM:SS`	UDF[FIRST_CONTACT] = 2017-04-25 17:12:30
`UDF[AMOUNT_LABEL ] = value`	1-255	Integers only, no decimal points, signs or symbols	UDF[BALANCE] = 1100

Table: warning: UDF labels can be up to 28 characters in length. UDF labels cannot begin with a number.

CHAPTER 22

Predictive Response

Predictive Response is a mechanism that can be used by Kount merchants to submit test requests and receive back predictable RIS responses. This means that a merchant, in order to test RIS, can generate a particular request that is designed to provide one or more specific RIS responses and/or errors. The predictive response inquiries are not actual RIS inquiries, which means the data will never be submitted to the Kount internal database.

An example would be if a merchant wanted to submit a RIS request that would return the very specific responses `SCOR = 71`, `AUTO = E`, and `GEOX = CA`.

Predictive Responses are created using the UDF (User Defined Fields) override option. These User Defined Fields do not need to be created through the Agent Web Console, they can be simply passed in as additional fields in the Predictive Response RIS inquiry.

:warning: In order to create a Predictive Response RIS Inquiry, the request must contain a specific email parameter in the EMAL field: `predictive@Kount.com`.

All other elements of the RIS request you submit must be valid elements and contain the minimum set of required RIS keys.

The basic syntax is: `UDF[~K!_label]="foo"` `~K!_` is the prefix, `label` is the desired field for which you want a response, such as `SCOR` or `ERRO`, and after the equal sign (=), enter the specific value you want returned. The `~K!_` prefix is required to trigger the UDF to become a predictive response field.

- Example 1: You want to send in a request that will result in a Kount Score of `18`, an Auto Decision of `E`, and a `601 System Error` code.

Request:

```
UDF[~K!_SCOR]=18
UDF[~K!_AUTO]=E
UDF[~K!_ERRO]=601
```

Response:

```
SCOR=18
AUTO=E
ERRO=601
```

- Example 2: You want to pass in a request that will result in a Kount Score of `42`, an Auto Decision of `Decline` and a `GEOX` of `Nigeria`.

Request:

```
UDF[~K!_SCOR]=42
UDF[~K!_AUTO]=D
UDF[~K!_GEOX]=NG
```

Response:

```
SCOR=42
AUTO=D
GEOX=NG
```

You can use UDF overrides to pass in an unlimited number of mock requests but all of the fields you pass in that are not overrides must be valid. In the response, all of the other elements, besides the UDF overrides will be the default values, including `MODE` and `MERC`.

CHAPTER 23

Kount Ris Data Collector

23.1 Kount Data Collector

The *Kount Data Collector* runs in the background at a sub second level while the user is logging into the website via a web clients or browser (see section below) or via a mobile app (`iOS or Android`).

Here are the standard requirements for any Data Collection event.

Note: Access Inquiry Service is designed to be used in conjunction with the Data Collection process. The Data Collection process is passive and provides no information back to a merchant independent of the Access Inquiry Service.

23.2 Session ID Discussion

The Session ID is the identifier for the collection event and is specific to the user's request. You will use the Session ID for subsequent calls to the API service for device information regarding the current user's interaction.

- Data Collector should be run once for each user's session within the web browser.
- Session ID field `name = sessionId`
- Session ID values must be exactly 32 character length and must be alpha-numeric values `(0-9, a-z or A-Z)`. Dashes `(-)` and underscores `(_)` are acceptable.
- Session IDs must be unique per request. They must be unique forever, they may not be recycled.
- Script tag parameter `value = s` Example: `s=abcdefg12345abababab123456789012.`

23.3 Web Clients or Browser

The Data Collector runs on a client's browser and collects a variety of information that helps uniquely identify the device.

Add the `<script>` tag and an `<img>` tag to the web page where you want to trigger the `Data Collection` to occur.

Field	Parameter	Value
merchantId	`m`	six digit Merchant ID number issued by Kount
sessionId	`s`	32 character session id; see Session ID Discussion above for more information

. Below is an example for the sandbox02 environment where the Merchant ID field `(m=123456)` and the Session ID field `(s=abcdefg12345abababab123456789012)` are set.

```
<script type='text/javascript' src='https://sandbox02.kaxsdc.com/collect/sdk?␣
↪m=123456&s=abcdefg12345abababab123456789012'> </script>
<img src='https://sandbox02.kaxsdc.com/logo.gif?m=123456&
↪s=abcdefg12345abababab123456789012' />
```

Note: The script tag will not affect the UI with its placement. The logo.gif is a 1x1 pixel transparent image served by Kount. This is preset image that is set by Kount within the Data Collection process.

23.4 Creating the kaxsdc Class (Responsible for Triggering Data Collection)

The Client Collector SDK allows the data collection process to be triggered by any number of events. The default event is page load. These events can be configured by adding the kaxsdc class and `data-event='<event>'` to an HTML element. The kaxsdc class is configurable using the Client Collector SDK.

23.5 Namespace & ka.ClientSDK Object

All `Kount collector JavaScript` is namespaced under the `ka JavaScript` object. To start using the Client Collector SDK, create a new ClientSDK object: `var client=new ka.ClientSDK();`

Available methods in the ka.ClientSDK object:

Method	Description
className	Sets the class to be used by `autoLoadEvents()`
autoLoadEvents()	Automatically load events to trigger the data collection process. This will wire all elements with a class equal to the property className that also have a data-event attribute After the first event fires and the data collection process begins, no further events will have an effect.
collectData()	Manually initiates the data collection process instead of waiting for an event to be loaded using the autoLoadEvents() method.
setupCallback(config)	A client programmable callback system that allows the client to execute custom code at certain points in the data collection process. This method allows a merchant to add a callback function to be called at a specified life-cycle hook. A merchant can pass a JavaScript object containing one or more life cycle hooks with a function pointer or an anonymous function to be executed. List of hooks (in order of firing): • `collect-begin` - Triggers when the collection starts. • `collect-end` - Triggers when the collection ends. When executed, the callback function is passed a JavaScript object containing the following properties: • `MercSessId` - The merchant provided session. • `MerchantId` - The merchant Id.

23.5.1 Code Example:

This code will fire an alert when the process reaches the `collect-begin` hook

```
<html>
   .
   .
   .
   <body class='kaxsdc' data-event='load'>
      .
      .
      .
      <script type='text/javascript'>
        var client=new ka.ClientSDK();
        client.setupCallback(
            {
                // fires when collection has finished
                'collect-end':
                    function(params) {
                        // enable login button
                        loginButton = document.getElementById('login_button');
                        loginButton.removeAttribute('disabled');
                        // now user can login and navigate away from the page
                    },
                // fires when collection has started
```

(continues on next page)

(continued from previous page)

```
                'collect-begin':
                    function(params) {
                        // add hidden form element to post session id
                        var loginForm = document.forms['loginForm'];
                        var input = document.createElement('input');
                        input.type = 'hidden';
                        input.name = 'kaId';
                        input.value = params['MercSessId'];
                        loginForm.appendChild(input);
                    }
            }
        );
        // The auto load looks for the default, an element with the 'kaxsdc' class and
        // data-event equal to a DOM event (load in this case). Data collection begins
        // when that event fires on that element--immediately in this example
        client.autoLoadEvents();
      </script>
   </body>
</html>
```

23.5.2 Alternative Integration Example

For maximum efficiency in data collection, initiating data collection when the body loads is best. However, if your use-case demands that data collection is initiated by a different event, then this example may be helpful.

```
<html>
    <body>
        <button class='mycustomclass' data-event='click'>Click Me!</button>
      <script type='text/javascript'>
        var client=new ka.ClientSDK();
        // notice the use of the custom class
        client.className = 'mycustomclass';
        client.autoLoadEvents();
      </script>
    </body>
</html>
```

23.5.3 Another Optional Example to use if you would rather not wait, then just call collectData()

```
<html>
    <body>
      <script type='text/javascript'>
        var client=new ka.ClientSDK();
        client.setupCallback(
            {
                // fires when collection has finished
                'collect-end':
                    function(params) {
                    location.href = 'http: //example.com/loginpage';
                    }
            }
```

(continues on next page)

(continued from previous page)

```
        );
        client.collectData();
      </script>
    </body>
</html>
```

23.6 SDK for Native Mobile Apps (iOS and Android)

The implementation of the Client Collector is somewhat different for native Mobile Apps. Kount has a native Mobile SDK for both iOS and Android which is compatible with both the Kount Complete and Kount Access products. By using the native Mobile SDK, along with a Merchant ID, Session ID, and custom URL for posting, native mobile apps can take advantage of the added capabilities from these native SDKs. These native Mobile SDKs collect more data and increase the reliability of more consistent fingerprint across the life of a device.

The Data Collector SDK for Android provides a java jar file which can be used to perform Device Collection interaction with Kount for native Android applications.

- For Android implementations see the Android SDK Guide
- For iOS implementation see the iOS SDK Guide

Note: The Access Inquiry Service is designed to be used in conjunction with the Data Collection process. The Data Collection process is passive and provides no information back to a merchant independent of the Access Inquiry Service.

CHAPTER 24

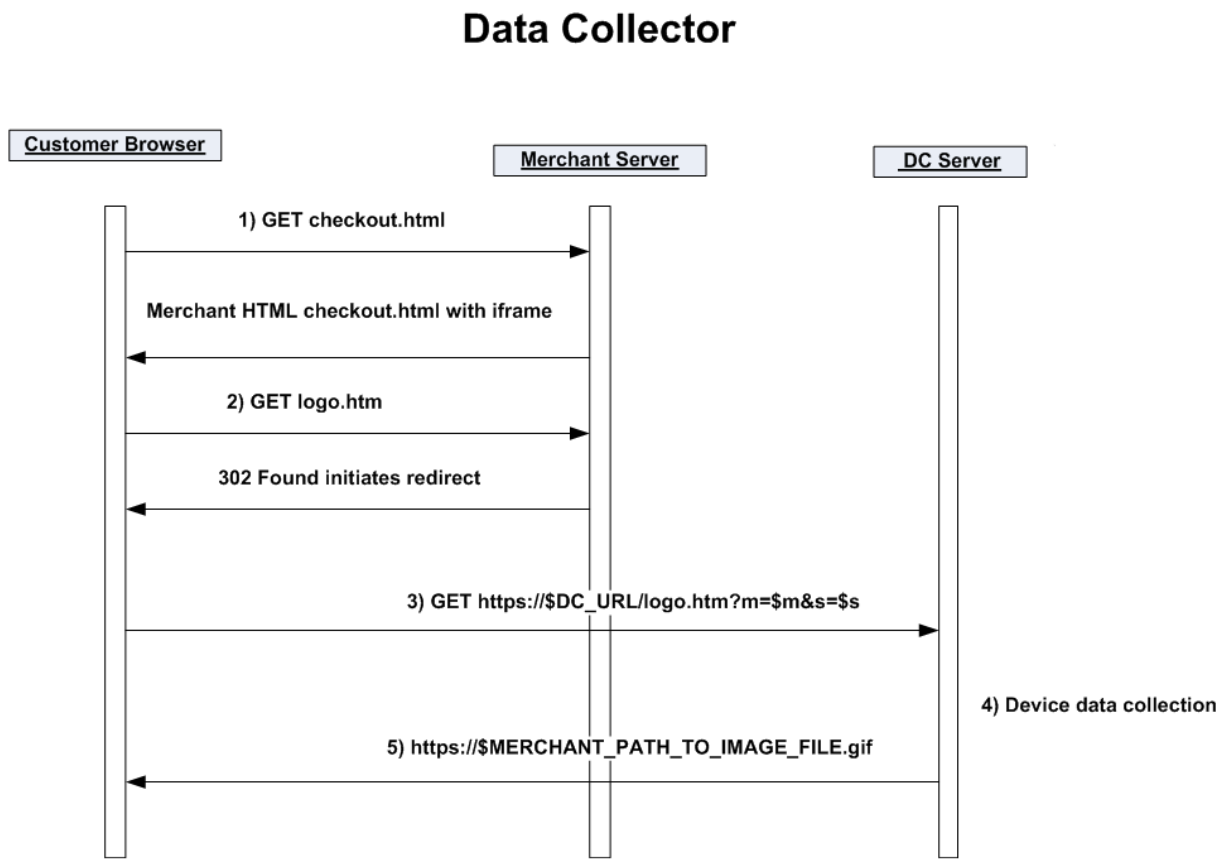
Data Collector

Quick link: * *Data Collector FAQ and Troubleshooting*

CHAPTER 25

Feature description

The Data Collector process can be used to obtain data related to the device initiating the customer transaction. This process is transparent to the customer and sub second. The data collected is used in conjunction with the RIS data.



The following sequence describes the Data Collector process: 1. Customer browses to merchant order form page

containing Data Collector code 2. Customer browser automatically requests redirect on the Data Collector element 3. Customer browser is redirected to Kount servers 4. Kount collects device attributes 5. Kount directs customer browser to display static image hosted by merchant

CHAPTER 26

Requirements

1. Port 443 must be available to post and receive data from Kount.
2. Merchant adds code to their checkout process.
    - HTML iframe:
        - The iframe should be placed on the order form page where the payment method or credit card information is requested, usually near the bottom of the page.
        - Server side code:
            - The server side code consists of a logo.htm and logo.gif server side executable scripts.
            - The path to the server side code must be a fully qualified path.
        - Code to create a unique session identifier:
            - When the form page is accessed, a session identifier must be created and stored to be used as the session identifier in the subsequent RIS post from the merchant to Kount. This identifier must be unique for at least 30 days and must be unique for every transaction submitted by each unique customer. If a single session ID were to be used on multiple transactions, those transactions would link together and erroneously affect the persona information and score.
3. Merchant supplies static image URL to Kount.
    - The static image can be an image that is currently being displayed on the page or Kount can provide an image if desired.
    - If the path or image requires change by the merchant subsequent to moving into production, Kount must be notified of the new path or filename to avoid possible failure.

CHAPTER 27

Sample code

Although a bit ancient (.jsp lol), the following code example demonstrates how to code for the data collector. Additional details can be found in the *Data Collector FAQ and Troubleshooting* page.

> :warning: All of the code presented here is for example purposes only. Any production implementation performed by the customer should follow all internal code quality standards of the customer's organization including security review.

```
<!--
  HTML iframe example
  Requirements - The iframe has a minimum width=1 and height=1
-->

<iframe width=1 height=1 frameborder=0 scrolling=no src='https://MERCHANT_URL/logo.
↪htm?m=merchantId&s=sessionId'>
    <img width=1 height=1 src='https://MERCHANT_URL/logo.gif?m=merchantId&s=sessionId
↪'>
</iframe>
```

CHAPTER 28

Phone-to-Web order submissions

When a merchant submits phone orders via the same web page interface as a customer, the data regarding the merchant's device is being sent to Kount, not the customer's device data. This will cause order linking to occur and in time will elevate the score of all orders associated with the persona.

:warning: Linking will also occur if the browser session ID is used as the transaction session ID and multiple orders are submitted from within the same browser session without closing the browser. To avoid this type of linking, the browser session ID could be incremented appending the UNIX timestamp, choose a different methodology for creating the session ID, or agents must close the browser between orders to ensure a new session has been created.

There are two different methods for receiving phone orders.

1. If the customer service agents navigate to a separate order entry page that does not collect iframe data: Call Center/Phone Orders will be posted as a Mode=P; hard code the IP address specifically to 10.0.0.1 and provide the phone number within the ANID field (if no phone number is available, pass 0123456789 hard coded).
2. If the customer service agents navigate to the same page as the customer (iframe data is collected): don't perform the redirect to the Kount Data Collector service, just post Call Center Orders as a Mode=Q; hard code the IP address specifically to 10.0.0.1.

In any of the above circumstances, if the email address is not provided to the agents, the agents will need to input noemail@Kount.com as the email address in order to prevent linking.

CHAPTER 29

Data Collector FAQ and Troubleshooting

Quick links

CHAPTER 30

General

Q: Where do I get the Merchant ID?

A Sandbox Boarding Information document will be sent following the initial kick-off call with the Merchant ID and URLs associated with the DC and RIS processes. A separate document for production will be sent with the production service URLs once the test transaction have been certified.

Q: Why does Kount require a redirect?

The redirect gathers device information without interfering with the order process. The redirect also obfuscates the communication between Kount and the customer.

Q: How do I receive a login to the AWC?

Kount will create the initial administrator user. Once the user has been created an automated e-mail will be sent requesting a password creation.

Q: Should I send production traffic to the test environment to test with?

Production traffic should not be sent to the test environment due to the possibility of skewed scoring from the test orders.

CHAPTER 31

Iframe

Q: Where should I place the iframe on my website?
If multiple forms of payment methods are available, i.e. Credit Card, PayPal, Bill Me Later, the iframe should be included on the page where the payment method is chosen. If only a single payment method is used, i.e. Credit Card, then the iframe should be included on the page where the customer inputs their credit card information.

:warning: Place the iframe only on one page. Do not place the image on multiple form pages on your website.

Q: Are there size restrictions on the iframe?
Yes, the iframe must be at least 1x1 pixels in size.

Q: Does it matter where the iframe shows up on the web page?
No, this is left to the merchant to decide.

Q: Why an iframe?
If a user were to source the page there would be no mention of Kount in the source. It also eliminates the possibility of being indexed by various search engines or crawlers.

Q: Why are there two files in the iframe?
The logo.gif file is a fallback in case iframes have been disabled or the browser does not support iframes.

Q: Are both the iframe and image required?
Yes both are required to ensure that a connection is made by the customer device.

Q: Why don't I just use the image, we don't have any iframes anywhere else on our site.

The manner in which iframes are handled by browsers provides greater insight to Kount.

Q: Why do the .htm and .gif files get interpreted as server side code?
By using .htm and .gif file extensions there is less concern from end users that may inspect the source code.

CHAPTER 32

Session Identifier

Q: What does the session identifier do?

The session identifier is used to join the device data with the order data sent by the RIS process. When the RIS process posts data to Kount it must use the session identifier that was created when the customer initiated the DC HTML.

Q: Does the session identifier need to be unique?

Yes, the session identifier must be unique over a thirty day time period. If there are duplicate session identifiers, device data originating from the DC process may be erroneously connected to RIS order data.

Q: Are there limitations on the session identifier?

Yes, it must be alpha-numeric with a minimum of one character and a maximum of 32 characters long.

Q: What should I use for the session identifier?

The merchant determines, as long as the limitation guidelines are followed. Many merchants use a portion of the application session that is generated when the page is created as the session identifier.

Q: What happens when a user leaves the page after a session identifier has been created then returns to finish the order?

There can be multiple session identifiers created, only the last session will be used to join with the RIS transaction.

CHAPTER 33

Image

Q: Does it matter where the image is displayed on the page?
It is recommended to display the iframe below the fold of the initial web page but can be located anywhere on the page.

Q: Why does the merchant need to give Kount the path to an image for the redirection?
Kount must have this path to finish the 302 redirect for the image to load. If the path has not been supplied to Kount a broken image icon will be displayed on the merchant page.

Q: Does the image need to be accessible via the Internet?
Yes the image must be publicly available from the Internet.

Q: Why does the path need to be HTTPS?
If the image is not secure, a notification will appear alerting the customer that there are unsecure items on the check out form.

Q: Does it matter what the name of the file is?
If a customer were to inspect the source of the page there should be no indication of interaction from Kount. To alleviate the possibility of a merchant's customer questioning the interaction between Kount and the merchant do not include any reference to Kount in the file name.

Q: Do you have an example of an image?
Yes, it can be provided upon request. Please contact your Client Success Manager for further details. See example below (Secure Payments):


Secure
Payments

CHAPTER 34

Troubleshooting

Q: Is the correct Kount URL being used?
Verify that the correct Kount Data Collector URL is being used, test URL or production URL.

Q: Have you provided Kount with the HTTPS URL path to the image?
If the URL path has not been set there will be a broken image displayed on the page.

Q: Is the image available via the Internet?
Test this by pasting the path of the URL in a browser on an external network and verify that the image appears.

Q: Have appropriate DNS entries, NATs, and firewall settings been configured correctly?
Due to the security concerns regarding test environments or production environment the merchant's network operations may need to verify that proper access is available.

Q: Are the logo.htm and logo.gif files being interpreted as server-side code?
If the files are not interpreted as server-side code, when requested the files will serve up the source code instead of performing the redirect. This can be tested by pointing the browser directly to the logo.htm or logo.gif URLs and verify that the static image appears. If source code appears, then the files are not being interpreted correctly. This can also be tested via a UNIX wget command.

Q: Does the redirect contain the correct Merchant ID?
Verify that the redirect Merchant ID is the correct six digit ID supplied by Kount.

Q: Is the Session ID created in the DC process the same session ID being sent with the RIS post?
Ensure that the Session ID being created and stored during the DC process is the correct one being used in the RIS post to Kount and adheres to the session ID requirements.

Q: Only part of the device data is collected, Javascript, Time Zone and other details seem to be missing?
The logo.gif server side script is calling the log.gif instead of the logo.htm. See the *Data Collector* - Server Side Code section.

Q: Why do some of the items within the Extended Variables gadget not display or display as N/A?
A fully qualified path must be used within the scr value of the iFrame.

CHAPTER 35

Payment Types

The Kount RIS Python SDK defines a group of objects representing various payment types. Using those payment types with the `Request.set_payment(...)` method automatically sets the required `PTYP` parameter and other parameters corresponding to the selected payment type.

Supported payment types:

- `CardPayment`
- `CheckPayment`
- `GiftCardPayment`
- `GooglePayment`
- `GreenDotMoneyPakPayment`
- `PaypalPayment`
- `Apple Pay`
- `BPAY`
- `Carte Bleue`
- `ELV`
- `GiroPay`
- `Interac`
- `Mercado Pago`
- `Neteller`
- `POLi`
- `Single Euro Payments Area`
- `Skrill/Moneybookers`
- `Sofort`

- `Token`

There are also several "system" payment types:

- `NoPayment`
- `BillMeLaterPayment`

CHAPTER 36

Session related Parameters

There are a few parameters responsible for maintaining connection between linked interactions with the RIS. They are transported as a part of the `Request/Response` objects during standard RIS communication.

- `SESS` parameter This parameter should be created by the merchant at the start of each new customer purchase. `SESS` is used to join the customer device data with the order data sent with the RIS request. If the merchant uses the Kount [[Data Collector]] service to obtain customer device information, then the same `SESS` value must be used for all RIS calls starting with the one to the Data Collector service. Requirements for the parameter value are:
- alpha-numeric
- length: 1-32 characters
- value should be unique over a thirty-day period of time `SESS` is a mandatory parameter set by `Request.session_set(string)` method.
- `TRAN` parameter The `TRAN` parameter is required for `Update` calls to Kount RIS. Its value is created by Kount and is returned within the `Response` object for the first RIS `Inquiry`. For all subsequent events, modifying this particular customer order, the 'TRAN' parameter should be set to the Kount-created value.

An important use of the RIS response is the ability to verify if the decision-making process was successful and view any warnings or errors that were made during the RIS post from the merchant. All warnings will be displayed in the response and if errors do occur the RIS response will be returned with a `MODE = E`.

Here's a list of all used RIS warning and error codes.

CHAPTER 37

RIS Warning and Error Codes

Response Code	Warning/Error Label	Response Code Description
201	MISSING_VERS	Missing version of Kount, this is built into SDK but must be supplied by merchant if not using the SDK
202	MISSING_MODE	The mode type for post is missing. Refer to the Request Types
203	MISSING_MERC	The six digit Merchant ID was not sent
204	MISSING_SESS	The unique session ID was not sent
205	MISSING_TRAN	Transaction ID number
211	MISSING_CURR	The currency was missing in the RIS submission
212	MISSING_TOTL	The total amount was missing
221	MISSING_EMAL	The email address was missing
222	MISSING_ANID	For MODE = P RIS inquiries the caller ID is missing
223	MISSING_SITE	The website identifier that was created in the Agent Web Console (DEFAULT is the default website ID) is missing
231	MISSING_PTYP	The payment type is missing. Refer to the RIS Payment Types
232	MISSING_CARD	The credit card information is missing
233	MISSING_MICR	Missing Magnetic Ink Character Recognition string
234	MISSING_PYPL	The PayPal Payer ID is missing
235	MISSING_PTOK	The payment token is missing.

Continued on next page

Table 1 – continued from previous page

Response Code	Warning/Error Label	Response Code Description
241	MISSING_IPAD	The IP address is missing
251	MISSING_MACK	The merchant acknowledgement is missing
261	MISSING_POST	The RIS query submitted to Kount contained no data
271	MISSING_PROD_TYPE	The shopping cart data array attribute is missing.
272	MISSING_PROD_ITEM	The shopping cart data array attribute is missing.
273	MISSING_PROD_DESC	The shopping cart data array attribute is missing.
274	MISSING_PROD_QUANT	The shopping cart data array attribute is missing.
275	MISSING_PROD_PRICE	The shopping cart data array attribute is missing.
301	BAD_VERS	The version of Kount supplied by merchant does not fit the four integer parameter
302	BAD_MODE	The mode type is invalid. Refer to the RIS Inquiry Service Modes: inquiry
303	BAD_MERC	The six digit Merchant ID is malformed or wrong
304	BAD_SESS	The unique session ID is invalid. Refer to the Data Collector
305	BAD_TRAN	Transaction ID number is malformed
311	BAD_CURR	The currency was wrong in the RIS submission
312	BAD_TOTL	The total amount is wrong. TOTL is the whole number amount charged to customer
321	BAD_EMAL	The email address does not meet required format or is greater than 64 characters in length
322	BAD_ANID	For MODE = P RIS inquiries the caller ID is malformed
323	BAD_SITE	The website identifier that was created in the Agent Web Console (DEFAULT is the default w website ID) does not match what was created in the AWC.
324	BAD_FRMT	The specified format is wrong. Format options are key value pairs, XML, JSON, YAML
331	BAD_PTYP	The payment type is wrong. Refer to the RIS Payment Types
332	BAD_CARD	The credit card information is malformed or wrong, test cards do not work in the production environment

Continued on next page

Table 1 – continued from previous page

Response Code	Warning/Error Label	Response Code Description
333	BAD_MICR	Malformed or improper Magnetic Ink Character Recognition string.
334	BAD_PYPL	The PayPal Payer ID is malformed or corrupt.
335	BAD_GOOG	Malformed or improper Google Checkout Account ID string.
336	BAD_BLML	Malformed or improper Bill Me Later account number.
337	BAD_PENC	The encryption method specified is wrong.
338	BAD_GDMP	The GreenDot payment token is not a valid payment token
339	BAD_HASH	When payment type equals CARD, PTYP = CARD and payment encryption type equals KHASH, `PENC = KHASH the value must be 20 characters in length.
340	BAD_MASK	Invalid or excessive characters in the PTOK field
341	BAD_IPAD	The IP address does not match specifications
342	BAD_GIFT	The Gift Card payment token is invalid due to invalid characters, null, or exceeding character length
351	BAD_MACK	The merchant acknowledgement must be Y or N
362	BAD_CART	There is a discrepancy in the shopping cart key count and the number of items actually being sent in the cart
371	BAD_PROD_TYPE	The shopping cart data array attribute is missing.
372	BAD_PROD_ITEM	The shopping cart data array attribute is corrupt or missing.
373	BAD_PROD_DESC	The shopping cart data array attribute is corrupt or missing.
374	BAD_PROD_QUANT	The shopping cart data array attribute is corrupt or missing.
375	BAD_PROD_PRICE	The shopping cart data array attribute is corrupt or missing.
399	BAD_OPTN	A UDF has been mistyped or does not exist in the Agent Web Console
401	EXTRA_DATA	RIS keys submitted by merchant were not part of SDK
404	UNNECESSARY_PTOK	When PTYP equals NONE and a PTOK is submitted
413	REQUEST_ENTITY_TOO_LARGE	The RIS Post to Kount exceeded the 4K limit.
501	UNAUTH_REQ	Error regarding certificate - Using test certificate in prod

Continued on next page

Table 1 – continued from previous page

Response Code	Warning/Error Label	Response Code Description
502	UNAUTH_MERC	Invalid Merchant ID has been entered
601	SYS_ERR	Unspecified system error - Contact Merchant Services
602	SYS_NOPROCESS	Kount will not process particular transaction
701	NO_HDR	No header found with merchantId = [XXXXX], session_id = [htot2kk5khpamo45f777q455], trans=[122347] This error occurs when a RIS request goes to the database and there is no data available in the reply. The Update post had an invalid transaction ID#. Check all required fields for update post and confirm they are being passed correctly.

- **Missing**: When this designation appears, the customer has failed to complete a required field.
- **Bad**: When this designation appears, some data was sent but failed to meet specifications. This could also be explained as malformed data or bad code that did not meet specifications, such as AVS = W instead of AVS = M.

CHAPTER 38

kount package

38.1 kount.client module

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"""class Client"""

from __future__ import (absolute_import, unicode_literals,
                        division, print_function)

import logging
import requests

from . import config
from . import request
from . import response
from .version import VERSION
from .config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


LOG = logging.getLogger('kount.client')


class Client:
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, api_url, api_key,
                 timeout=None,
                 raise_errors=None):
        """
        Client class used to execute Kount request
        :param api_url: endpoint to which the request should be sent
        :param api_key: merchant api key, provided by Kount
        :param timeout: request timeout, if not set the default value
         from SDKConfig will be used
        :param raise_errors: indicates if request validation error should
         be thrown, if not set the default value from SDKConfig will be used
        """
        self.api_url = api_url
        self.api_key = api_key
        conf = config.SDKConfig
        self.timeout = timeout or conf.get_default_timeout()
        LOG.debug("url - %s, len_key - %s", self.api_url, len(self.api_key))

    def process(self, req):

        if not isinstance(req, request.Request):
            raise ValueError("invalid request, %s" % type(request))

        res = self._execute(req.params)
        if res:
            return response.Response(res)
        return None

    def _execute(self, params):
        """validate data and request post
        https://pypi.python.org/pypi/requests
        Use simplejson if available.
        if raise_errors==False, the validation errors will not be raised,
        only logged; by default raise_errors=True"""
        if LOG.isEnabledFor(logging.DEBUG):
            for key, param in params.items():
                LOG.debug('%s=%s', key, param)

        headers_api = {'X-Kount-Api-Key': self.api_key}

        if self.api_key == "API_KEY":
            raise Exception('Please configure the API Key in settings.py file')

        params['FRMT'] = 'JSON'
        LOG.debug("url=%s, headers=%s, params=%s", self.api_url,
                  headers_api, params)

        req = requests.post(self.api_url,
                            headers=headers_api,
                            data=params,
                            timeout=self.timeout)

        try:
            resp = req.json()
        except ValueError as jde:
            LOG.error("ValueError - %s", jde)
            try:
```

(continues on next page)

(continued from previous page)

```
                text_to_json = parse_k_v(req.text)
                LOG.debug("process text: %s", text_to_json)
                return text_to_json
            except ValueError:
                error = "Neither JSON nor String %s" % req.text
                LOG.debug(error)
                raise ValueError(error)
        else:
            LOG.debug("MERC = %s, SESS = %s, SDK ELAPSED = %s ms.",
                      params.get('MERC', None),
                      params.get("SESS", None),
                      req.elapsed.total_seconds())
            return resp


def parse_k_v(text):
    """parse text to dict"""
    return dict(c.split('=', 1) for c in text.split('\n'))
```

38.2 kount.inquiry module

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved
"""RIS initial inquiry class"""
import time

import ipaddress
import logging
from datetime import datetime

from .config import SDKConfig
from .request import (Request, CurrencyType, InquiryMode,
                      Gender, AddressType, ShippingType)
from .util.address import Address
from .util.cartitem import CartItem
from .version import VERSION

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS

# common logger for inquiry and request
LOG = logging.getLogger('kount.request')


class Inquiry(Request):
    """RIS initial inquiry class.
        Contains specific methods for setting various inquiry properties
        Class constructor. Sets the RIS mode to "Inquiry" ("Q"),
```

(continues on next page)

(continued from previous page)

```
        sets currency to "USD", and sets the Python SDK identifier.
        The mode and currency can be adjusted by called
        InquiryMode and CurrencyType methods respectively.
        """
    def __init__(self):
        super(Inquiry, self).__init__()
        self.version()
        self.params["SDK"] = SDKConfig.SDK
        self.params["ANID"] = ""
        self.params["FRMT"] = "JSON"
        self.params["VERS"] = SDKConfig.VERS
        self.inquiry_mode = InquiryMode.DEFAULT
        self.currency_type = CurrencyType.USD
        LOG.debug('Inquiry: %s', self.params)

    def version(self):
        """SDK_Type-RIS_VERSION-SDK_BUILD_DATETIMESTAMP"""
        datestr = datetime.now().strftime('%Y%m%d%H%M')
        vers = "Sdk-Ris-%s-%s" % (SDKConfig.LANG, SDKConfig.SDK_VERSION)
        assert len(vers) <= 32
        self.set_param("SDK_VERSION", vers)

    def set_cash(self, cash):
        """Set cash amount of any feasible goods.
            Arg: cash - int, cash amount of any feasible goods"""
        self.set_param("CASH", cash)

    def set_date_of_birth(self, dob):
        """Set the date of birth in the format YYYY-MM-DD.
         Arg: dob - Date of birth
         """
        self.set_param("DOB", dob)

    def set_gender(self, gender):
        """Set the gender. Either M(ale) of F(emale).
            Acceptable values: GENDER
            Arg: - gender"""

        self.set_param("GENDER", gender)

    def set_user_defined_field(self, label, value):
        """Set a user defined field.
            Arg: label - The name of the user defined field
            Arg: value - The value of the user defined field
        """
        self.set_param("UDF[%s]" % label, value)

    def set_request_mode(self, mode):
        """Set the request mode.
            Acceptable values are: InquiryMode
            Arg: mode - Mode of the request
        """
        self.set_param("MODE", mode)


    def set_currency(self, currency):
        """Set the three character ISO-4217 currency code.
```

(continues on next page)

(continued from previous page)

```
            Arg: currency - Type of currency, CurrencyType
        """
        self.set_param("CURR", currency)

    def set_total(self, total=0):
        """Set the total amount in lowest possible denomination of currency.
            Arg: total - Transaction amount in lowest possible
            denomination of given currency
        """
        self.set_param("TOTL", total)

    def set_email_client(self, email_addr):
        """Set the email address of the client.
            Arg: email - Email address of the client
        """
        self.set_param("EMAL", email_addr)

    def set_customer_name(self, c_name):
        """the name of the client or company.
            Arg: c_name - Name of the client or company
         """
        self.set_param("NAME", c_name)

    def _address(self, adr_type, address):
        """Set the address.
            Arg: address - The billing or shipping address; type Address
            adr_type - billing or shipping, values in ['B', 'S']
        """
        if not isinstance(address, Address):
            raise ValueError
        self.params[adr_type + "2A1"] = address.address1
        self.params["%s2A2" % adr_type] = address.address2
        self.params["%s2CI" % adr_type] = address.city
        self.params["%s2ST" % adr_type] = address.state
        self.params["%s2PC" % adr_type] = address.postal_code
        self.params["%s2CC" % adr_type] = address.country
        self.params["%sPREMISE" % adr_type] = address.premise
        self.params["%sSTREET" % adr_type] = address.street

    def set_billing_address(self, address):
        """Set the billing address.
            Arg: address - The billing address, type Address
            Address
        """
        LOG.debug("B2A1 = %s, B2A2 = %s, B2CI = %s, B2ST = %s, "
                  "B2PC = %s, B2CC = %s, BPREMISE = %s, BSTREET = %s",
                  address.address1, address.address2, address.city,
                  address.state, address.postal_code, address.country,
                  address.premise, address.street)
        self._address(AddressType.BILLING, address)

    def set_shipping_address(self, address):
        """Set the shipping address.
            Arg: address - The shipping address, type Address
        """
        LOG.debug("S2A1 = %s, S2A2 = %s, S2CI = %s, S2ST = %s, "
                  "S2PC = %s, S2CC = %s, SPREMISE = %s, SSTREET = %s",
```

(continues on next page)

(continued from previous page)

```
                address.address1, address.address2, address.city,
                address.state, address.postal_code, address.country,
                address.premise, address.street)
        self._address(AddressType.SHIPPING, address)

    def set_billing_phone_number(self, billing_phone=""):
        """Set the billing phone number.
            Arg: billing_phone - Billing phone number
         """
        self.set_param("B2PN", billing_phone)

    def set_shipping_phone_number(self, shipping_phone):
        """Set the shipping phone number.
            Arg: shipping_phone - shipping phone number
         """
        self.set_param("S2PN", shipping_phone)

    def set_shipping_name(self, ship_name=""):
        """Set the shipping name.
            Arg: ship_name - Shipping name
        """
        self.set_param("S2NM", ship_name)

    def set_email_shipping(self, shipping_email):
        """Set the shipping email address of the client.
            Arg: shipping_email - shipping email
        """
        self.set_param("S2EM", shipping_email)

    def set_unique_customer_id(self, unique_customer):
        """Set the unique ID or cookie set by merchant.
           Arg: unique_customer - Customer-unique ID or cookie set by merchant.
        """
        self.set_param("UNIQ", unique_customer)

    def set_ip_address(self, ip_adr):
        """Set the IP address. ipaddress
        Arg: ip_adr - IP Address of the client
        """
        ipaddress.ip_address(ip_adr)
        self.set_param("IPAD", ip_adr)

    def set_user_agent(self, useragent):
        """Set the user agent string of the client.
        Arg: useragent - user agent string of the client
        """
        self.set_param("UAGT", useragent)

    def set_timestamp(self, time_stamp=None):
        """Set the timestamp (in seconds) since the UNIX epoch
            for when the UNIQ value was set.
            Arg: time_stamp -  The timestamp
        """
        if time_stamp is None:
            time_stamp = time.time()
        self.set_param("EPOC", time_stamp)
```

(continues on next page)

(continued from previous page)

```
    def set_shipment_type(self, shipment):
        """Set shipment type
            Arg: shipment -  type ShippingType
        """

        self.set_param("SHTP", shipment)

    def set_anid(self, anid_order):
        """Set the anid
            Automatic Number Identification (ANI) submitted with order.
            If the ANI cannot be determined,
            merchant must pass 0123456789 as the ANID.
            This field is only valid for MODE=P RIS submissions.
            Arg: anid_order - Anid of the client
        """
        self.set_param("ANID", anid_order)

    def set_company_name(self, name):
        """Set the name of the company.
        Arg: name - Name of the company
        """
        self.set_param("NAME", name)

    def set_website(self, web_site):
        """Set the website.
            Arg: site - the website
        """
        self.set_param("SITE", web_site)

    def set_shopping_cart(self, cart):
        """Set the shopping cart.
            Arg: cart - Cart items in the shopping cart, type Cart
        """
        for index, item in enumerate(cart):
            if not isinstance(item, CartItem):
                raise ValueError('Invalid cart item: %s', item)
                LOG.debug("PROD_TYPE[%i] = %s, PROD_ITEM[%i] = %s, "
                      "PROD_DESC[%i] = %s, PROD_QUANT[%i] = %s, "
                      "PROD_PRICE[%i] = %s",
                      index, item.product_type,
                      index, item.item_name,
                      index, item.description,
                      index, item.quantity,
                      index, item.price)

            self.params["PROD_TYPE[%i]" % index] = item.product_type
            self.params["PROD_ITEM[%i]" % index] = item.item_name
            self.params["PROD_DESC[%i]" % index] = item.description
            self.params["PROD_QUANT[%i]" % index] = item.quantity
            self.params["PROD_PRICE[%i]" % index] = item.price
```

38.3 kount.request module

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"""RIS Request superclass for Inquiry and Update"""

import logging

from .config import SDKConfig
from .util.khash import Khash
from .util import payment as payments
from .version import VERSION

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS

LOG = logging.getLogger('kount.request')


class _RisType(object):

    __CACHED_ATTRS = None

    @classmethod
    def is_valid(cls, val):
        attrs = cls.__CACHED_ATTRS
        if attrs is None:
            attrs = {
                v for k, v in vars(cls).items() if not k.startswith('_')
            }
            cls.__CACHED_ATTRS = attrs
        return val in attrs


class AuthStatus(_RisType):
    """Authorization status"""
    APPROVE = 'A'
    DECLINE = 'D'
    REVIEW = 'R'
    ESCALATE = "E"
    REVIEW_TIMEOUT = 'X'
    APPROVED_DECLINED_TIMEOUT = 'Y'
    ELEVATED = 'C'


class BankcardReply(_RisType):
    """Bankcard Reply"""
    MATCH = 'M'
    NO_MATCH = 'N'
    UNAVAILABLE = 'X'
```

(continues on next page)

(continued from previous page)

```
class Gender(_RisType):
    """gender"""
    MALE = 'M'
    FEMALE = 'F'


class AddressType(_RisType):
    """address type"""
    BILLING = 'B'
    SHIPPING = 'S'


class ShippingType(_RisType):
    """
    "SD". Same day shipping type.
    "ND". Next day shipping type.
    "2D". Second day shipping type.
    "ST". Standard shipping type.
    """
    SAME_DAY = 'SD'
    NEXT_DAY = 'ND'
    SECOND_DAY = '2D'
    STANDARD = "ST"


class RefundChargebackStatus(_RisType):
    """Refund charge back status.
    R - The transaction was a refund.
    C - The transaction was a chargeback.
    """
    REFUND = 'R'
    CHARGEBACK = 'C'


class MerchantAcknowledgment(_RisType):
    """merchant acknowledgment
    "Y". The product expects to ship.
    "N". The product does not expect to ship.
    """
    FALSE = 'N'
    TRUE = 'Y'


class CurrencyType(_RisType):
    """Currency type object
    "USD". United States Dollars
    "EUR". European currency unit
    "CAD". Canadian Dollar
    "AUD". Australian Dollar
    "JPY". Japanese Yen
    "HKD". Honk Kong Dollar
    "NZD". New Zealand Dollar
    """
    USD = 'USD'
    EUR = 'EUR'
    CAD = 'CAD'
```

(continues on next page)

(continued from previous page)

```
    AUD = 'AUD'
    JPY = 'JPY'
    HKD = 'HKD'
    NZD = 'NZD'


class InquiryMode(_RisType):
    """
    "Q". Default inquiry mode, internet order type.
    "P". Phone order type.
    "W". Kount Central Mode W - Full Inquiry [W]ith thresholds.
    "J". Kount Central Mode J - Fast Inquiry [J]ust thresholds.
    """
    DEFAULT = 'Q'
    PHONE = 'P'
    WITH_THRESHOLDS = 'W'
    JUST_THRESHOLDS = 'J'


class Request(object):
    """RIS Request superclass for Inquiry and Update."""

    def __init__(self):
        """Map containing data that will be sent to RIS."""
        self.params = dict()
        self.payment = None
        self.close_on_finish = None

    def set_param(self, key, value):
        """Set a parm for the request.
        Args:
           key - The key for the parm
           value - The value for the parm
        """
        self.params[key] = value
        LOG.debug("%s = %s", key, value)

    def set_khash_payment_encoding(self, enabled=True):
        """Set KHASH payment encoding.
        Arg: enabled Boolean
        """
        self.set_param("PENC", "KHASH" if enabled else "")

    def set_version(self, version):
        """Set the version number.
        Args: version - The SDK version
        """
        self.set_param("VERS", version)

    def set_lbin(self, lbin):
        """Set the  Long Bank Identification Number.
        Arg: lbin - string
        """
        self.set_param("LBIN", lbin)

    def set_session_id(self, session_id):
        """Set the session id. Must be unique over a 30-day span
```

(continues on next page)

(continued from previous page)

```
        Args: session_id -  Id of the current session
        """
        self.set_param("SESS", session_id)

    def set_merchant(self, merchant_id):
        """Set the merchant id.
        Args: merchant_id - Merchant ID
        """
        self.set_param("MERC", merchant_id)

    def set_kount_central_customer_id(self, customer_id):
        """Set the Kount Central Customer ID.
        Args: customer_id - KC Customer ID
        """
        self.set_param("CUSTOMER_ID", customer_id)

    def set_order_number(self, order_number):
        """Set the order number.
        Args: order_number - Merchant unique order number
        """
        self.set_param("ORDR", order_number)

    def set_merchant_acknowledgment(self, ma_type):
        """Set the merchant acknowledgment.
        Merchants acknowledgement to ship/process the order.
        The MACK field must be set as MerchantAcknowledgment.TRUE
        if personal data is to be
        collected to strengthen the score.
        Args: ma_type - merchant acknowledgment type
        """
        if MerchantAcknowledgment.is_valid(ma_type):
            self.set_param("MACK", ma_type)
        else:
            raise ValueError("Invalid MerchantAcknowledgment = %s" % ma_type)

    def set_authorization_status(self, auth_status):
        """Set the Authorization Status.
        Authorization Status returned to merchant from processor.
        Acceptable values for the
        AUTH field are AuthStatus. In orders where AUTH=A will
        aggregate towards order velocity of the persona while
        orders where AUTH=D will
        decrement the velocity of the persona.
        Args: auth_status - Auth status by issuer
        """
        if AuthStatus.is_valid(auth_status):
            self.set_param("AUTH", auth_status)
        else:
            raise ValueError("Invalid AuthStatus value %s" % auth_status)

    def set_avs_zip_reply(self, avs_zip_reply):
        """Set the Bankcard AVS zip code reply.
        Address Verification System Zip Code verification response
        returned to merchant from
        processor. Acceptable values are BCRSTAT.
        Args: avs_zip_reply - Bankcard AVS zip code reply
        """
```

(continues on next page)

(continued from previous page)

```
        if BankcardReply.is_valid(avs_zip_reply):
            self.set_param("AVSZ", avs_zip_reply)
        else:
            raise ValueError('Invalid BankcardReply = %s' % avs_zip_reply)

    def set_avs_address_reply(self, avs_address_reply):
        """Set the Bankcard AVS street addres reply.
        Address Verification System Street verification response
        returned to merchant from processor. Acceptable values are BCRSTAT.
        Args: avs_address_reply - Bankcard AVS street address reply
        """
        if BankcardReply.is_valid(avs_address_reply):
            self.set_param("AVST", avs_address_reply)
        else:
            raise ValueError('Invalid BankcardReply = %s' % avs_address_reply)

    def set_avs_cvv_reply(self, cvv_reply):
        """Set the Bankcard CVV/CVC/CVV2 reply.
        Card Verification Value response returned to merchant from processor.
        Acceptable values are BCRSTAT
        Args: cvv_reply -  Bankcard CVV/CVC/CVV2 reply
        """
        if BankcardReply.is_valid(cvv_reply):
            self.set_param("CVVR", cvv_reply)
        else:
            raise ValueError('Invalid BankcardReply = %s' % cvv_reply)

    def set_payment(self, payment):
        """ Set a payment.
            Depending on the payment type, various request parameters are set:
            PTOK, PTYP, LAST4.
            If payment token hashing is not possible, the PENC parameter is set
            to empty string.
            Args: payment -  Payment
        """
        khasher = Khash.get()
        if "PENC" in self.params \
                and not isinstance(payment, payments.NoPayment) \
                and not payment.khashed:
            try:
                if not self.params.get("MERC"):
                    raise ValueError("merchant_id not set")
                if isinstance(payment, payments.GiftCardPayment):
                    merchant_id = int(self.params["MERC"])
                    payment.payment_token = khasher.hash_gift_card(
                        merchant_id, payment.payment_token)
                else:
                    payment.payment_token = khasher.hash_payment_token(
                        payment.payment_token)
                payment.khashed = True
                self.set_param("PENC", "MASK")
                LOG.debug("payment.khashed=%s", payment.khashed)
            except ValueError as nfe:
                LOG.debug("Error converting Merchant ID to integer"
                          " value. Set a valid Merchant ID. %s",
                          str(nfe))
                raise nfe
```

(continues on next page)

(continued from previous page)

```
            except Exception as nsae:
                LOG.debug("Unable to create payment token hash. Caught %s "
                          "KHASH payment encoding disabled", str(nsae))
                # Default to plain text payment tokens
                self.params["PENC"] = ""
        if khasher.khashed(payment.payment_token):
            self.set_param("PENC", "KHASH")
        self.set_param("PTOK", payment.payment_token)
        self.set_param("PTYP", payment.payment_type)
        self.set_param("LAST4", payment.last4)

    @staticmethod
    def _mask_token(token):
        """Encodes the provided payment token according to the MASK
           encoding scheme
           Args: token -  the Payment token for this request
           return - MASK-encoded token
        """
        encoded = token[0:6]
        for _ in range(6, len(token) - 4, 1):
            encoded.append('X')
        encoded.append(token[-4:])
        LOG.debug("mask_token = %s", token)
        return encoded

    def set_payment_by_type(self, ptyp, ptok):
        """ Set a payment by payment type and payment token.
        The payment type parameter provided is checked
        if it's one of the predefined payment types
        and Payment is created appropriately
        Args: ptyp - See SDK documentation for a list of accepted payment types
        ptok - The payment token
        """
        cls = {
            "BLML": payments.BillMeLaterPayment,
            'CARD': payments.CardPayment,
            'CHECK': payments.CheckPayment,
            'GIFT': payments.GiftCardPayment,
            'GOOG': payments.GooglePayment,
            'GDMP': payments.GreenDotMoneyPakPayment,
            'NONE': payments.NoPayment,
            'PYPL': payments.PaypalPayment,
        }.get(ptyp)
        if cls is None:
            cls = payments.Payment
        self.set_payment(cls(ptyp, ptok))

    def set_masked_payment(self, payment):
        """ Sets a card payment and masks the card number in the following way:
        First 6 characters remain as they are, following characters up to the
        last 4 are replaced with the 'X' character, last 4 characters
        remain as they are.
        If the provided Payment parameter is not a card payment,
        standard encoding will be applied.
        This method sets the following RIS Request fields:
        PTOK, PTYP, LAST4, PENC.
        Args: payment - card payment
```

(continues on next page)

(continued from previous page)

```
        """
        token = payment.payment_token
        if isinstance(self.payment, payment.CardPayment) and \
                not payment.khashed:
            token = self._mask_token(token)
            self.set_param("PTOK", token)
            self.set_param("PTYP", payment.payment_type)
            self.set_param("LAST4", payment.last4)
            self.set_param("PENC", "MASK")
        else:
            self.set_param("PTOK", token)
            LOG.debug("Payment Masked: provided payment is not "
                      "a CardPayment, applying khash instead of masking")
            self.set_payment_by_type(payment, token)

    def is_set_khash_payment_encoding(self):
        """Check if KHASH payment encoding has been set.
           return boolean TRUE when set.
        """
        encoded = self.params.get("PENC") == "KHASH"
        LOG.debug("is_set_khash_payment_encoding = %s", encoded)
        return encoded

    def set_close_on_finish(self, close_on_finish):
        """Set a flag for the request transport.
           Arg: close_on_finish - Sets the close_on_finish flag
           return boolean TRUE when set.
        """
        self.close_on_finish = close_on_finish
        LOG.debug("close_on_finish = %s", close_on_finish)


class UpdateMode(_RisType):
    """UpdateMode - U, X"""
    NO_RESPONSE = 'U'
    WITH_RESPONSE = 'X'


class Update(Request):
    """RIS update class.
     defaults to update_mode WithResponse.
     """

    def __init__(self):
        super(Update, self).__init__()
        self.set_mode(UpdateMode.NO_RESPONSE)
        self.params['VERS'] = SDKConfig.VERS
        # self.params["SDK"] = "python"
        self.set_khash_payment_encoding(True)

    def set_mode(self, mode):
        """Set the mode.
        Args - mode - Mode of the request
        """
        if UpdateMode.is_valid(mode):
            self.params["MODE"] = mode
        else:
```

(continues on next page)

(continued from previous page)

```
            raise ValueError("Invalid UpdateMode: %s" % mode)

    def set_transaction_id(self, transaction_id):
        """Set the transaction id.
        Arg - transaction_id, String Transaction id
        """
        self.params["TRAN"] = transaction_id

    def set_refund_chargeback_status(self, rc_status):
        """Set the Refund/Chargeback status: R = Refund C = Chargeback.
        Arg - rc_status, String Refund or chargeback status
        """
        if RefundChargebackStatus.is_valid(rc_status):
            self.params["RFCB"] = rc_status
        else:
            raise ValueError("Invalid RefundChargebackStatus: %s" % rc_status)
```

38.4 kount.response module

38.5 kount.ris_validator module

38.6 kount.settings module

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.

import os

from . import resources
from .version import VERSION
from .settings import CONFIGURATION_KEY, SDK_AUTHOR, SDK_MAINTAINER, MAINTAINER_EMAIL,
↪ DEFAULT_TIMEOUT

__author__ = SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDK_MAINTAINER
__email__ = MAINTAINER_EMAIL
__status__ = "Development"


class SDKConfig:

    VERS = "0720"

    SDK = "PYTH"

    LANG = "Python"
```

(continues on next page)

(continued from previous page)

```
    SDK_VERSION = "0.0.0"

    SDK_AUTHOR = __author__

    SDK_MAINTAINER = __maintainer__

    MAINTAINER_EMAIL = __email__

    STATUS = __status__
    # default behaviour for request failures
    _RAISE_ERRORS = True

    # requests timeout
    _DEFAULT_TIMEOUT = DEFAULT_TIMEOUT

    # should be set from the sdk user
    _CONFIGURATION_KEY = CONFIGURATION_KEY

    @classmethod
    def get_default_timeout(cls):
        return cls._DEFAULT_TIMEOUT

    @classmethod
    def get_configuration_key(cls):
        return cls._CONFIGURATION_KEY

    @classmethod
    def get_should_raise_validation_errors(cls):
        return cls._RAISE_ERRORS

    @classmethod
    def setup(cls,
              config_key,
              default_timeout=5,
              raise_errors=True,
              xml_rules_file_name=None):
        """
        Call this method before start using the SDK
        :param config_key: mandatory parameter, configuration key provided
         by Kount
        :param default_timeout: request timeout, default value is 5 seconds
        :param raise_errors: indicate if the request should throw an exception
         in case of error, default value is True
        :param xml_rules_file_name: xml rules for validation of the request,
         should not be overwritten, unless you know what you are doing
        """

        cls._CONFIGURATION_KEY = config_key
        cls._DEFAULT_TIMEOUT = default_timeout
        cls._RAISE_ERRORS = raise_errors

        from .util import khash
        k = khash.Khash(config_key)
        k.verify()
```

38.7 kount.version module

```
# Set the value to a VERSION in this dedicated module in the project
VERSION = '3.3.3'
```

38.8 Subpackages

38.8.1 kount util

kount.util.address module

kount.util.cartitem module

kount.util.khash module

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"class Khash"

from __future__ import (absolute_import, unicode_literals,
                        division, print_function)

import hashlib
import logging
import re
import string

from kount.resources.correct_key_cryp import correct_key_cryp
from kount.version import VERSION
from .a85 import a85decode
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS

LOG = logging.getLogger('kount.khash')


def validator(*args):
    """check is list with arguments is valid -
    positive integer or length of the string
    Args: list with integers or strings
    returns: list with args as strings
    """
    for current in args:
        try:
            str(current)
```

(continues on next page)

(continued from previous page)

```
            is_string = True
        except ValueError:
            is_string = False
        curr_len = len(str(current))
        invalid = curr_len < 6 or curr_len > 100 or current is None
        try:
            current = int(current)
            is_digit = True
        except (TypeError, ValueError):
            is_digit = False
        if is_digit and int(current) <= 0:
            raise ValueError("incorrect arg: [%s]" % current)
        elif invalid or not is_string:
            raise ValueError("incorrect arg: [%s]" % current)
    return [str(i) for i in args]


class Khash(object):

    re_khashed = re.compile(r"^[0-9a-zA-Z]{6}[0-9A-Z]{14}$")
    __instance = None

    @staticmethod
    def get():
        """ Static access method. """
        SDKConfig.setup(SDKConfig._CONFIGURATION_KEY)

        return Khash.__instance

    """
    Class for creating Kount RIS KHASH encoding payment tokens.
    """
    def __init__(self, key):
        """ Virtually private constructor. """
        hash_salt_key = a85decode(key)
        self.config_key = hash_salt_key.decode("utf-8")
        self.verify()
        self.salt_key = hash_salt_key
        Khash.__instance = self

    def verify(self):
        key = self.config_key.encode('utf-8')
        sha_key = hashlib.sha256(key).hexdigest()
        if sha_key != correct_key_cryp:
            mesg = "Configured config_key key is incorrect"
            LOG.error(mesg)
            raise ValueError(mesg)

        LOG.info("Configured config_key is correct.")
        return True

    def hash_payment_token(self, token):
        """Create a Kount hash of a provided payment token. Payment tokens
        that can be hashed via this method include: credit card numbers,
        Paypal payment IDs, Check numbers, Google Checkout IDs, Bill Me
        Later IDs, and Green Dot MoneyPak IDs.
```

(continues on next page)

(continued from previous page)

```
        Args: token - String to be hashed
        returns: String hashed
        if len(token) < 6 - need to clarify the expected behaviour
        """
        token_valid = validator(token)[0]
        return "%s%s" % (token_valid[:6], self.hash(plain_text=token_valid))

    def hash_gift_card(self, merchant_id, card_number):
        """ Hash a gift card payment token using the Kount hashing algorithm.
        Args: merchant_id - Merchant ID number
        card_number - Card number to be hashed
        returns: String hashed
        """
        merchant_id, card_number = validator(merchant_id, card_number)
        return "%s%s" % (merchant_id, self.hash(plain_text=card_number))

    def hash(self, plain_text):
        """
        Compute a Kount hash of a given plain text string.
        Preserves the first six characters of the input
        so that hashed tokens can be categorized
        by Bank Identification Number (BIN).
        Args: plain_text - String to be hashed
        returns: String hashed
        """
        if isinstance(plain_text, int):
            plain_text = str(plain_text)
        if validator(plain_text):
            legal_chars = string.digits + string.ascii_uppercase
            loop_max = 28
            hex_chunk = 7
            length = len(legal_chars)
            hashed = []
            plain_text_bytes = plain_text.encode('utf-8')  # Python 3.x
            sha1 = hashlib.sha1(plain_text_bytes + ".".encode('utf-8') +
                                self.salt_key).hexdigest()
            for i in range(0, loop_max, 2):
                hashed.append(legal_chars[int(sha1[i: i + hex_chunk], 16)
                                          % length])
            return ''.join(hashed)
        else:
            raise ValueError("incorrect arg: [%s]" % plain_text)

    @classmethod
    def khashed(cls, val):
        """ Arg: val - String, Token that may or may not be khashed
         return: Boolean, True if token is already khashed
        """
        return cls.re_khashed.match(val)
```

kount.util.payment module

```
#!/usr/bin/env python
"class Payment - RIS payment type object"
# -*- coding: utf-8 -*-
```

(continues on next page)

(continued from previous page)

```
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.

from kount.util.khash import Khash
from kount.version import VERSION
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


class Payment(object):
    """RIS payment type object.
    Args:
    payment_type - Payment type.
    payment_token - Payment token
    khashed - Indicates whether payment token is khashed.
    True if payment token is khashed.
    """
    def __init__(self, payment_type=None, payment_token=None, khashed=True):
        """Constructor for a payment that accepts the payment ID.
        Calculate and set the payment token LAST4 value.
        last4 - Last 4 characters of payment token"""
        self.last4 = "NONE"
        self.payment_token = None
        if payment_type is not None:
            self._payment_type = str(payment_type)
        else:
            self._payment_type = "NONE"

        if payment_token is not None:
            self.payment_token = str(payment_token)
            if len(self.payment_token) >= 4:
                self.last4 = self.payment_token[-4:]
            if khashed:
                self.khashed = self.khash_token()
        self.khashed = khashed and Khash.khashed(self.payment_token)

    @property
    def payment_type(self):
        return self._payment_type

    def khash_token(self):
        "hash the payment_token, return True if khashed, else raise ValueError"
        k = Khash.get()
        self.payment_token = k.hash_payment_token(
            token=self.payment_token)
        if k.khashed(self.payment_token):
            return True
        raise ValueError("payment_token [%s] is not khashed" %
                         self.payment_token)
```

(continues on next page)

(continued from previous page)

```
class GiftCardPayment(Payment):
    """Sets the PTYP parameter to GIFT,
    params: gift_card_number,
     khashed - boolean"""
    def __init__(self, gift_card_number, khashed=True):
        super(GiftCardPayment, self).__init__(
            payment_type="GIFT",
            payment_token=gift_card_number,
            khashed=khashed)


class GooglePayment(Payment):
    """Sets the PTYP parameter to "GIFT".
    params: google_payment_id - Google payment ID
                khashed - boolean"""
    def __init__(self, google_payment_id, khashed=True):
        super(GooglePayment, self).__init__(
            payment_type="GOOG",
            payment_token=google_payment_id,
            khashed=khashed)


class GreenDotMoneyPakPayment(Payment):
    """Sets the PTYP parameter to "GDMP".
    params: green_dot_mp_payment - Green Dot MoneyPak payment ID number
            khashed - boolean"""
    def __init__(self, green_dot_mp_payment, khashed=True):
        super(GreenDotMoneyPakPayment, self).__init__(
            payment_type="GDMP",
            payment_token=green_dot_mp_payment,
            khashed=khashed)


class NoPayment(Payment):
    """No payment type. Sets the PTYP parameter to "NONE", not khashed"""
    def __init__(self, *args, **kwargs):
        super(NoPayment, self).__init__(
            payment_type=None,
            payment_token=None,
            khashed=False)


class CheckPayment(Payment):
    """Sets the PTYP parameter to "CHEK".
    params: micr - The MICR (Magnetic Ink Character Recognition)
                line on the check.
            khashed - boolean
    """
    def __init__(self, micr, khashed=True):
        super(CheckPayment, self).__init__(
            payment_type="CHEK",
            payment_token=micr,
            khashed=khashed)


class PaypalPayment(Payment):
    """paypal payment - accepts the paypal payment ID.
```

(continues on next page)

(continued from previous page)

```
    Sets the PTYP parameter to "PYPL".
    params: paypal_payment_id - Paypal payment ID
            khashed - boolean
    """
    def __init__(self, paypal_payment_id, khashed=True):
        super(PaypalPayment, self).__init__(
            payment_type="PYPL",
            payment_token=paypal_payment_id,
            khashed=khashed)


class CardPayment(Payment):
    """credit card payment
    Sets the PTYP parameter to "CARD".
    params: card_number - The card number
            khashed - boolean
    """

    def __init__(self, card_number, khashed=True):
        super(CardPayment, self).__init__(
            payment_type="CARD",
            payment_token=card_number,
            khashed=khashed)


class BillMeLaterPayment(Payment):
    """bill me later payment
    Sets the PTYP parameter to "BLML".
    params: payment_id - The payment ID,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(BillMeLaterPayment, self).__init__(
            payment_type="BLML",
            payment_token=payment_id,
            khashed=khashed)


class ApplePay(Payment):
    """Apple Pay
    Sets the PTYP parameter to "APAY".
    params: payment_id - apay,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(ApplePay, self).__init__(
            payment_type="APAY",
            payment_token=payment_id,
            khashed=khashed)


class BPayPayment(Payment):
    """BPay Payment
    Sets the PTYP parameter to "BPAY".
    params: payment_id - bpay,
            khashed - boolean
    """
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, payment_id, khashed=True):
        super(BPayPayment, self).__init__(
            payment_type="BPAY",
            payment_token=payment_id,
            khashed=khashed)


class CarteBleuePayment(Payment):
    """Carte Bleue Payment
    Sets the PTYP parameter to "CARTE_BLEUE".
    params: payment_id - Carte Bleue id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(CarteBleuePayment, self).__init__(
            payment_type="CARTE_BLEUE",
            payment_token=payment_id,
            khashed=khashed)


class ELVPayment(Payment):
    """ELV Payment
    Sets the PTYP parameter to "ELV".
    params: payment_id - ELV id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(ELVPayment, self).__init__(
            payment_type="ELV",
            payment_token=payment_id,
            khashed=khashed)


class GiroPayPayment(Payment):
    """GIROPAY Payment
    Sets the PTYP parameter to "GIROPAY".
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(GiroPayPayment, self).__init__(
            payment_type="GIROPAY",
            payment_token=payment_id,
            khashed=khashed)


class InteracPayment(Payment):
    """Interac Payment
    Sets the PTYP parameter to "INTERAC".
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(InteracPayment, self).__init__(
            payment_type="INTERAC",
            payment_token=payment_id,
            khashed=khashed)
```

(continues on next page)

(continued from previous page)

```
class MercadoPagoPayment(Payment):
    """Mercado Pago Payment
    Sets the PTYP parameter to "MERCADE_PAGO".
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(MercadoPagoPayment, self).__init__(
            payment_type="MERCADE_PAGO",
            payment_token=payment_id,
            khashed=khashed)


class NetellerPayment(Payment):
    """Neteller Payment
    Sets the PTYP parameter to "NETELLER".
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(NetellerPayment, self).__init__(
            payment_type="NETELLER",
            payment_token=payment_id,
            khashed=khashed)


class PoliPayment(Payment):
    """POLi Payment
    Sets the PTYP parameter to "POLI".
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(PoliPayment, self).__init__(
            payment_type="POLI",
            payment_token=payment_id,
            khashed=khashed)


class SEPAPayment(Payment):
    """Single Euro Payments Area Payment
    Sets the PTYP parameter to "SEPA".
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(SEPAPayment, self).__init__(
            payment_type="SEPA",
            payment_token=payment_id,
            khashed=khashed)


class SofortPayment(Payment):
    """Sofort Payment
    Sets the PTYP parameter to "SOFORT".
```

(continues on next page)

(continued from previous page)

```
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(SofortPayment, self).__init__(
            payment_type="SOFORT",
            payment_token=payment_id,
            khashed=khashed)


class TokenPayment(Payment):
    """Token Payment
    Sets the PTYP parameter to "TOKEN".
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(TokenPayment, self).__init__(
            payment_type="TOKEN",
            payment_token=payment_id,
            khashed=khashed)


class SkrillPayment(Payment):
    """Skrill/Moneybookers Payment
    Sets the PTYP parameter to "SKRILL".
    params: payment_id - id,
            khashed - boolean
    """
    def __init__(self, payment_id, khashed=True):
        super(SkrillPayment, self).__init__(
            payment_type="SKRILL",
            payment_token=str(payment_id),
            khashed=khashed)


def NewPayment(payment_type, payment_token, khashed=True):
    """User-defined payment type
    Sets the PTYP parameter to desired parameter.
    params: payment_type
            payment_token
            khashed - boolean
    """
    return Payment(payment_type, payment_token, khashed=khashed)
```

kount.util.risexception module

kount.util.validation_error module

kount.util.xmlparser module

38.8.2 kount resources

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"correct configurationKey - sha-256"

from kount.version import VERSION
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


correct_key_cryp = "ce1638f9cf6ce31b4ead12ca49712f1e538a055a6152ec83c7285e56ce917bd6"
```

38.9 Tests

38.9.1 Unit and Integration tests

conftest.py

```
import os
import pytest

from kount import config


def pytest_addoption(parser):
    parser.addoption('--conf-key', action='store',
                     default=os.environ.get('CONF_KEY', ''))
    parser.addoption('--api-key', action='store',
                     default=os.environ.get('RIS_SDK_SANDBOX_API_KEY', ''))
    parser.addoption('--merchant-id', action='store',
                     default=os.environ.get('RIS_SDK_SANDBOX_MERCHANT_ID', ''))
    parser.addoption('--api-url', action='store',
                     default=os.environ.get('RIS_SDK_SANDBOX_URL', 'https://risk.test.
↪kount.net'))


@pytest.fixture(scope='session', autouse=True)
def conf_key(request):
    try:
        config.SDKConfig.setup(request.config.getoption('--conf-key'))
    except ValueError as e:
        if not config.SDKConfig.get_configuration_key():
            msg = "Configuration key not set, use --conf-key or " \
```

(continues on next page)

(continued from previous page)

```
                    "set environment variable CONF_KEY"
        else:
            msg = 'Configuration key error: %s' % str(e)
        pytest.exit(msg)


@pytest.fixture(scope='class')
def api_key(request):
    request.cls.api_key = request.config.getoption('--api-key')


@pytest.fixture(scope='class')
def merchant_id(request):
    request.cls.merchant_id = request.config.getoption('--merchant-id')


@pytest.fixture(scope='class')
def api_url(request):
    request.cls.api_url = request.config.getoption('--api-url')
```

test_address.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
import unittest
from kount.util.address import Address
from kount.version import VERSION
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


class TestAddress(unittest.TestCase):
    """Address class test cases"""

    def test_address_valid(self):
        """valid address"""
        adr = Address(address1="567 West S2A1 Court North",
                      address2=None, state="Gnome", postal_code="AK",
                      premise="99762", country="US")
        self.assertTrue(isinstance(adr, Address))
        adr = Address("1234 North B2A1 Tree Lane South", None,
                      "Albuquerque", "NM", "87101", "US")
        self.assertTrue(isinstance(adr, Address))
        adr = Address("567 West S2A1 Court North", None,
                      "Gnome", "AK", "99762", "US")
        self.assertEqual("567 West S2A1 Court North", str(adr.address1))

    def test_address_incorrect_string(self):
```

(continues on next page)

(continued from previous page)

```
        """incorrect address"""
        for bad_type in [42**42, "<script>alert(42)</script>", None, "", 42]:
            adr = Address(bad_type)
            self.assertEqual("", str(adr.country))

    def test_address_cyrillic(self):
        """incorrect address - cyrillic"""
        for bad_type in ["", "'%=:*-+<", ""]:
            adr = Address(bad_type)
            self.assertEqual("", str(adr.country))
            self.assertEqual(bad_type, adr.address1)


if __name__ == "__main__":
    unittest.main()
```

test_api_kount.py

```
#!/usr/bin/env python
"""Test class TestAPIRIS"""
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
import logging
import unittest

import pytest

from kount.client import Client
from kount.version import VERSION

from .json_test import example_data_products
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS

LOGGER = logging.getLogger('kount')

expected1 = {
    'AUTO': 'R',
    'BRND': None,
    'BROWSER': None,
    'CARDS': '1',
    'COOKIES': None,
    'COUNTERS_TRIGGERED': 0,
    'COUNTRY': None,
    'DDFS': None,
    'DEVICES': '1',
    'DEVICE_LAYERS': '....',
    'DSR': None,
    'EMAILS': '1',
    'FINGERPRINT': None,
```

(continues on next page)

(continued from previous page)

```
    'FLASH': None,
    'GEOX': 'US',
    'HTTP_COUNTRY': None,
    'IP_CITY': None,
    'IP_COUNTRY': None,
    'IP_IPAD': None,
    'IP_LAT': None,
    'IP_LON': None,
    'IP_ORG': None,
    'IP_REGION': None,
    'JAVASCRIPT': None,
    'KAPT': 'N',
    'LANGUAGE': None,
    'LOCALTIME': ' ',
    'MERC': '',  # will be replaced
    'MOBILE_DEVICE': None,
    'MOBILE_FORWARDER': None,
    'MOBILE_TYPE': None,
    'MODE': 'Q',
    'NETW': 'N',
    'ORDR': 'F8E874A38B7B',
    'OS': None,
    'PC_REMOTE': None,
    'PIP_CITY': None,
    'PIP_COUNTRY': None,
    'PIP_IPAD': None,
    'PIP_LAT': None,
    'PIP_LON': None,
    'PIP_ORG': None,
    'PIP_REGION': None,
    'PREVIOUSLY_WHITELISTED': 'N',
    'PROXY': None,
    'REASON_CODE': None,
    'REGION': None,
    'REGN': None,
    'RULES_TRIGGERED': 1,
    'RULE_DESCRIPTION_0': 'Review if order total > $1000 USD',
    'SCOR': '34',
    'OMNISCORE':61.2,
    'SESS': 'F8E874A38B7B4B6DBB71492A584A969D',
    'SITE': 'DEFAULT',
    'THREE_DS_MERCHANT_RESPONSE': None,
    'TIMEZONE': None,
    'UAS': None,
    'VERS': '0710',
    'VOICE_DEVICE': None,
    'WARNING_COUNT': 0}


def dict_compare(dict1, dict2):
    """compare 2 dictionaries"""
    dict1_keys = set(dict1.keys())
    dict2_keys = set(dict2.keys())
    intersect_keys = dict1_keys.intersection(dict2_keys)
    added = dict1_keys - dict2_keys
    removed = dict2_keys - dict1_keys
    modified = {o: (dict1[o],
```

(continues on next page)

(continued from previous page)

```
                    dict2[o]) for o in intersect_keys if dict1[o] != dict2[o]}
    same = set(o for o in intersect_keys if dict1[o] == dict2[o])
    return added, removed, modified, same


CURLED = {
    'ANID': '',
    'AUTH': 'A',
    'AVST': 'M',
    'AVSZ': 'M',
    'B2A1': '1234 North B2A1 Tree Lane South',
    'B2CC': 'US',
    'B2CI': 'Albuquerque',
    'B2PC': '87101',
    'B2PN': '555+867-5309',
    'B2ST': 'NM',
    'CASH': '4444',
    'CURR': 'USD',
    'CVVR': 'M',
    'EMAL': 'curly.riscaller15@kountqa.com',
    'FRMT': 'JSON',
    'IPAD': '4.127.51.215',
    'LAST4': '2514',
    'MACK': 'Y',
    'MERC': '999666',
    'MODE': 'Q',
    'NAME': 'Goofy Grumpus',
    'ORDR': '088E9F496135',
    'PROD_DESC[]': '3000 CANDLEPOWER PLASMA FLASHLIGHT',
    'PROD_ITEM[]': 'SG999999',
    'PROD_PRICE[]': '68990',
    'PROD_QUANT[]': '2',
    'PROD_TYPE[]': 'SPORTING_GOODS',
    'PTOK': '0007380568572514',
    'PTYP': 'CARD',
    'S2A1': '567 West S2A1 Court North',
    'S2CC': 'US',
    'S2CI': 'Gnome',
    'S2EM': 'sdkTestShipTo@kountsdktestdomain.com',
    'S2NM': 'SdkTestShipToFirst SdkShipToLast',
    'S2PC': '99762',
    'S2PN': '208 777-1212',
    'S2ST': 'AK',
    'SESS': '088E9F4961354D4F90041988B8D5C66B',
    'SITE': 'DEFAULT',
    'TOTL': '123456',
    'UNIQ': '088E9F4961354D4F9004',
    'VERS': '0710'}


@pytest.mark.usefixtures("api_url", "api_key", "merchant_id")
class TestAPIRIS(unittest.TestCase):
    """
    implemented curl from https://kopana.atlassian.net/wiki/display/KS/Testing
    """
    maxDiff = None
```

(continues on next page)

(continued from previous page)

```
    timeout = 5

    def _expected_response(self):
        r = dict(expected1)
        r['MERC'] = self.merchant_id
        return r

    def test_api_kount(self):
        """expected modified 'TRAN'"""
        data = CURLED
        self.assertIn('MODE', CURLED)
        expected = {
            "VERS": "0710", "MODE": "Q", "TRAN": "PTPN0Z04P8Y6",
            "MERC": "999666", "SESS": "088E9F4961354D4F90041988B8D5C66B",
            "ORDR": "088E9F496135", "AUTO": "R", "SCOR": "29", "GEOX": "US",
            "BRND": None, "REGN": None, "NETW": "N", "KAPT": "N", "CARDS": "1",
            "DEVICES": "1", "EMAILS": "1", "VELO": "0",
            "VMAX": "0", "SITE": "DEFAULT", "DEVICE_LAYERS": "....",
            "FINGERPRINT": None, "TIMEZONE": None, "LOCALTIME": " ",
            "REGION": None,
            "COUNTRY": None, "PROXY": None, "JAVASCRIPT": None, "FLASH": None,
            "COOKIES": None, "HTTP_COUNTRY": None, "LANGUAGE": None,
            "MOBILE_DEVICE": None, "MOBILE_TYPE": None,
            "MOBILE_FORWARDER": None,
            "VOICE_DEVICE": None, "PC_REMOTE": None, "RULES_TRIGGERED": 1,
            "RULE_ID_0": "1024842",
            "RULE_DESCRIPTION_0": "Review if order total > $1000 USD",
            "COUNTERS_TRIGGERED": 0,
            "REASON_CODE": None, "DDFS": None, "DSR": None,
            "UAS": None, "BROWSER": None,
            "OS": None, "PIP_IPAD": None, "PIP_LAT": None, "PIP_LON": None,
            "PIP_COUNTRY": None,
            "PIP_REGION": None, "PIP_CITY": None, "PIP_ORG": None,
            "IP_IPAD": None,
            "IP_LAT": None, "IP_LON": None, "IP_COUNTRY": None,
            "IP_REGION": None,
            "IP_CITY": None, "IP_ORG": None, "WARNING_COUNT": 0,"OMNISCORE":None,
↪"PREVIOUSLY_WHITELISTED": "N", "THREE_DS_MERCHANT_RESPONSE":None}
        for raise_errors in [True, False]:
            actual = self._client(raise_errors=raise_errors)._execute(data)
            added, removed, modified, _ = dict_compare(actual, expected)
            self.assertEqual(added, set())
            self.assertEqual(removed, set())
            modified_exp = {
                'REGN': (actual['REGN'], expected['REGN']),
                'TRAN': (actual['TRAN'], expected['TRAN']),
                'SCOR': (actual['SCOR'], expected['SCOR']),
                'OMNISCORE': (actual['OMNISCORE'],  expected['OMNISCORE'])
                }
            self.assertEqual(sorted(modified), sorted(modified_exp))

    def test_api_kount_2_items(self):
        "expected modified 'TRAN'"
        data = example_data_products.copy()
        self.assertIn('MODE', data)
        for raise_errors in [True, False]:
            actual = self._client(raise_errors=raise_errors)._execute(data)
```

(continues on next page)

(continued from previous page)

```
            del (actual['TRAN'], actual['RULE_ID_0'],
                 actual['VELO'], actual['VMAX'])
            self.assertEqual(actual, self._expected_response())

    def test_last_2_items_bad_email(self):
        "last_2_items_bad_email"
        data = example_data_products.copy()
        self.assertIn('MODE', CURLED)
        bad = CURLED['EMAL'].replace('@', "%40")
        data["EMAL"] = bad

        expected = {
            'ERROR_0':
                "321 BAD_EMAL Cause: [[%s is an invalid email address],"
                " Field: [EMAL], Value: [%s]" % (bad, bad),
            'ERRO': 321,
            'ERROR_COUNT': 1,
            'WARNING_COUNT': 0,
            'MODE': 'E'}
        actual = self._client(raise_errors=False)._execute(data)
        self.assertEqual(actual, expected)

    def test_2_items_bad_s2em(self):
        """bad S2EM"""
        bad = example_data_products["S2EM"].replace('@', "%40")
        data = example_data_products.copy()
        data["S2EM"] = bad
        actual = self._client(raise_errors=False)._execute(params=data)
        del (actual['TRAN'], actual['RULE_ID_0'],
             actual['VELO'], actual['VMAX'])
        self.assertEqual(actual, self._expected_response())

    def test_two_items_none_email(self):
        "email = None"
        data = example_data_products.copy()
        data["EMAL"] = None
        self.assertIn('MODE', data)
        expected = {
            'ERRO': 221, 'ERROR_COUNT': 1,
            'MODE': 'E', 'WARNING_COUNT': 0,
            'ERROR_0': "221 MISSING_EMAL Cause: "
                       "[Non-empty value was required in this case], "
                       "Field: [EMAL], Value: []"}
        for raise_errors in [True, False]:
            actual = self._client(raise_errors=raise_errors)._execute(data)
            self.assertEqual(actual, expected)

    def test_two_items_missing_or_long_email(self):
        "missing or long incorrect email"
        data = example_data_products.copy()
        del data["EMAL"]
        self.assertIn('MODE', data)
        expected = {
            'ERRO': 221, 'ERROR_COUNT': 1,
            'MODE': 'E', 'WARNING_COUNT': 0,
            'ERROR_0': "221 MISSING_EMAL Cause: "
                       "[Non-empty value was required in this case], "
```

(continues on next page)

(continued from previous page)

```
                        "Field: [EMAL], Value: []"}
        for raise_errors in [True, False]:
            actual = self._client(raise_errors=raise_errors)._execute(data)
            self.assertEqual(actual, expected)
        data["EMAL"] = "a" * 57 + "@aaa.com"
        response = self._client(raise_errors=False)._execute(data)
        self.assertEqual(321, response['ERRO'])

    def test_api_kount_empty_data(self):
        "empty data"
        data = {'FRMT': 'JSON'}
        expected = {"MODE": "E", "ERRO": "201"}
        actual = self._client(raise_errors=False)._execute(data)
        self.assertEqual(actual, expected)

    def _client(self, **kwargs):
        kwargs['api_url'] = self.api_url
        kwargs['api_key'] = self.api_key
        kwargs['timeout'] = self.timeout
        return Client(**kwargs)
```

test_basic_connectivity.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"""Test Cases from sdk documentation
generate_unique_id
default_inquiry
Test Basic Connectivity
"""
import unittest
import pytest

from kount.client import Client
from kount.util.payment import CardPayment
from kount.version import VERSION

from .test_inquiry import generate_unique_id, default_inquiry

from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


PTOK = "0007380568572514"
EMAIL = 'predictive@kount.com'


@pytest.mark.usefixtures("api_url", "api_key", "merchant_id")
```

(continues on next page)

(continued from previous page)

```
class TestBasicConnectivity(unittest.TestCase):
    """Test Basic Connectivity"""
    maxDiff = None

    def _client(self, **kwargs):
        kwargs['api_url'] = self.api_url
        kwargs['api_key'] = self.api_key
        return Client(**kwargs)

    def _process(self, request, **client_kwargs):
        return self._client(**client_kwargs).process(request)

    def setUp(self):
        self.session_id = generate_unique_id()[:32]
        self.email_client = EMAIL
        payment = CardPayment(PTOK, khashed=False)
        self.inq = default_inquiry(self.merchant_id,
                                   self.session_id,
                                   self.email_client,
                                   payment=payment)

    def test_12_expected_score(self):
        "test_12_expected_score"
        self.inq.params["UDF[~K!_SCOR]"] = '42'
        res = self._process(self.inq)
        self.assertIsNotNone(res)
        self.assertEqual('42', res.get_score())

    def request_with_Lbin(self):
        "test_Lbin_set_in_requst"
        self.inq.params["LBIN"] = '1234567'
        self.assertEqual('1234567', self.inq.params.get("LBIN"))
        res = self._process(self.inq)
        self.assertEqual(0, len(res.get_errors))

    def request_without_Lbin(self):
        "test_Lbin_not_set_in_requst"
        res = self._process(self.inq)
        self.assertEqual(0, len(res.get_errors))

    def test_13_expected_decision(self):
        """test_13_expected_decision"""
        self.inq.params["UDF[~K!_AUTO]"] = 'R'
        res = self._process(self.inq)
        self.assertIsNotNone(res)
        self.assertEqual("R", res.get_auto())

    def test_16_expected_geox(self):
        """test_16_expected_geox"""
        self.inq.params["UDF[~K!_SCOR]"] = '42'
        self.inq.params["UDF[~K!_AUTO]"] = 'D'
        self.inq.params["UDF[~K!_GEOX]"] = 'NG'
        res = self._process(self.inq)
        self.assertIsNotNone(res)
        self.assertEqual("D", res.get_auto())
        self.assertEqual("NG", res.get_geox())
        self.assertEqual("42", res.get_score())
```

(continues on next page)

(continued from previous page)

```
    def test_cyrillic(self):
        """test_cyrillic"""
        bad = u' :№'
        self.inq.params["S2NM"] = bad
        self.inq.params["EMAL"] = bad
        res = self._process(self.inq, raise_errors=False)
        self.assertIsNotNone(res)
        actual = u"321 BAD_EMAL Cause: [[%s is an invalid email address]"\
                  ", Field: [EMAL], Value: [%s]" % (bad, bad)
        self.assertEqual({
            u'ERRO': 321,
            u'ERROR_0': actual,
            u'ERROR_COUNT': 1, u'MODE': u'E', u'WARNING_COUNT': 0},
            {u'ERRO':res.get_error_code(),
            u'ERROR_0': res.get_errors()[0],
            u'ERROR_COUNT': len(res.get_errors()),
            u'MODE': res.get_mode(),
            u'WARNING_COUNT': len(res.get_warnings())})

    def test_long(self):
        """test_long request"""
        bad_list = [
            ' :№',
            'abcqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq 12345']
        expected = """Neither JSON nor String """\
            """<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">\n"""\
            "<html><head>\n"\
            "<title>413 Request Entity Too Large</title>\n"\
            "</head><body>\n"\
            "<h1>Request Entity Too Large</h1>\n"\
            "The requested resource<br />/<br />\n"\
            "does not allow request data with POST requests, or the"\
            " amount of data provided in\n"\
            "the request exceeds the capacity limit.\n"\
            "</body></html>\n"\
            "MODE=E\n"\
            "ERRO=201"
        inq = self.inq
        for bad in bad_list:
            inq.params["S2NM"] = bad
            try:
                self._process(inq, raise_errors=False)
            except ValueError as vale:
                self.assertEqual(expected, str(vale))


class TestBasicConnectivityKhashed(TestBasicConnectivity):
    """Test Basic Connectivity Khashed"""
    maxDiff = None

    def setUp(self):
        self.session_id = generate_unique_id()[:32]
        self.email_client = EMAIL
        payment = CardPayment(PTOK)
        self.inq = default_inquiry(
            self.merchant_id, self.session_id,
```

(continues on next page)

(continued from previous page)

```
            self.email_client, payment=payment)


if __name__ == "__main__":
    unittest.main(
        # defaultTest="TestBasicConnectivity.test_16_expected_geox"
    )
```

json_test.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"example data from https://kopana.atlassian.net/wiki/display/KS/Testing"
from kount.version import VERSION
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


example_data = {
    'ANID': '',
    'AUTH': 'A',
    'AVST': 'M',
    'AVSZ': 'M',
    'B2A1': '1234+North+B2A1+Tree+Lane+South',
    'B2CC': 'US',
    'B2CI': 'Albuquerque',
    'B2PC': '87101',
    'B2PN': '555+867-5309',
    'B2ST': 'NM',
    'CASH': '4444',
    'CURR': 'USD',
    'CVVR': 'M',
    'EMAL': 'curly.riscaller15@kountqa.com',
    'FRMT': 'JSON',
    'IPAD': '4.127.51.215',
    'LAST4': '2514',
    'MACK': 'Y',
    'MERC': '999666',
    'MODE': 'Q',
    'NAME': 'Goofy+Grumpus',
    'ORDR': '088E9F496135',
    'PROD_DESC[]': '3000+CANDLEPOWER+PLASMA+FLASHLIGHT',
    'PROD_ITEM[]': 'SG999999',
    'PROD_PRICE[]': '68990',
    'PROD_QUANT[]': '2',
    'PROD_TYPE[]': 'SPORTING%5FGOODS',
    'PTOK': '0007380568572514',
    'PTYP': 'CARD',
    'S2A1': '567+West+S2A1+Court+North',
```

(continues on next page)

(continued from previous page)

```
    'S2CC': 'US',
    'S2CI': 'Gnome',
    'S2EM': 'sdkTestShipTo@kountsdktestdomain.com',
    'S2NM': 'SdkTestShipToFirst+SdkShipToLast',
    'S2PC': '99762',
    'S2PN': '208+777-1212',
    'S2ST': 'AK',
    'SESS': '088E9F4961354D4F90041988B8D5C66B',
    'SITE': 'DEFAULT',
    'TOTL': '123456',
    'UAGT': 'Mozilla%2F5.0+%28Macintosh%3B+Intel+Mac+OS+X+10%5F9%5F5%29+'\
            'AppleWebKit%2F537.36+%28KHTML%2C+like+Gecko%29+'\
            'Chrome%2F37.0.2062.124+Safari%2F537.36',
    'UNIQ': '088E9F4961354D4F9004',
    'VERS': '0710'
}

example_data_products = {
    'ANID': '',
    'AUTH': 'A',
    'AVST': 'M',
    'AVSZ': 'M',
    'B2A1': '1234+North+B2A1+Tree+Lane+South',
    'B2CC': 'US',
    'B2CI': 'Albuquerque',
    'B2PC': '87101',
    'B2PN': '555+867-5309',
    'B2ST': 'NM',
    'CASH': '4444',
    'CURR': 'USD',
    'CVVR': 'M',
    'EMAL': 'curly.riscaller15@kountqa.com',
    'FRMT': 'JSON', # set if not via sdk
    'IPAD': '129.173.116.98',
    'MACK': 'Y',
    'MERC': '999666',
    'MODE': 'Q',
    'NAME': 'Goofy+Grumpus',
    'ORDR': 'F8E874A38B7B',
    'PROD_DESC[0]': '3000+CANDLEPOWER+PLASMA+FLASHLIGHT',
    'PROD_DESC[1]': '3000+HP+NUCLEAR+TOILET',
    'PROD_ITEM[0]': 'SG999999',
    'PROD_ITEM[1]': 'TP999999',
    'PROD_PRICE[0]': '68990',
    'PROD_PRICE[1]': '1000990',
    'PROD_QUANT[0]': '2',
    'PROD_QUANT[1]': '44',
    'PROD_TYPE[0]': 'SPORTING%5FGOODS',
    'PROD_TYPE[1]': 'SPORTING%5FGOODS2',
    'PTOK': '0055071350519059',
    'PTYP': 'CARD',
    'S2A1': '567+West+S2A1+Court+North',
    'S2CC': 'US',
    'S2CI': 'Gnome',
    'S2EM': 'sdkTestShipTo@kountsdktestdomain.com',
    'S2NM': 'SdkTestShipToFirst+SdkShipToLast',
    'S2PC': '99762',
```

(continues on next page)

(continued from previous page)

```
    'S2PN': '208+777-1212',
    'S2ST': 'AK',
    'SESS': 'F8E874A38B7B4B6DBB71492A584A969D',
    'SITE': 'DEFAULT',
    'TOTL': '107783',
    'UAGT': 'Mozilla%2F5.0+%28Macintosh%3B+Intel+Mac+OS+X+10%5F9%5F5%29+'\
            'AppleWebKit%2F537.36+%28KHTML%2C+like+Gecko%29+'\
            'Chrome%2F37.0.2062.124+Safari%2F537.36',
    'UNIQ': 'F8E874A38B7B4B6DBB71',
    'SDK': 'PYTH',
    'VERS': '0710'
    }
```

test_inquiry.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"""Test Cases for Inquiry class"""
import pytest
import unittest
import uuid

from kount.client import Client
from kount.request import (AuthStatus, BankcardReply, InquiryMode,
                           CurrencyType, MerchantAcknowledgment)
from kount.inquiry import Inquiry
from kount.util.payment import CardPayment, Payment, GiftCardPayment
from kount.util.cartitem import CartItem
from kount.util.address import Address
from kount.config import SDKConfig
from kount.version import VERSION

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS

EMAIL_CLIENT = "sdkTest@kountsdktestdomain.com"
PTOK = "0007380568572514"
BILLING_ADDRESS = Address("1234 North B2A1 Tree Lane South",
                          "", "Albuquerque", "NM", "87101", "US")
SHIPPING_ADDRESS = Address("567 West S2A1 Court North", "",
                           "Gnome", "AK", "99762", "US")

expected = {
    'ANID': '',
    'AUTH': 'A',
    'AVST': 'M',
    'AVSZ': 'M',
    'B2A1': '1234 North B2A1 Tree Lane South',
    'B2A2': '',
    'B2CC': 'US',
    'B2CI': 'Albuquerque',
```

(continues on next page)

(continued from previous page)

```
    'B2PC': '87101',
    'B2PN': '555-867-5309',
    'B2ST': 'NM',
    'BPREMISE': '',
    'BSTREET': '',
    'CASH': '4444',
    'CURR': 'USD',
    'CVVR': 'M',
    'EMAL': EMAIL_CLIENT,
    'FRMT': 'JSON',
    # 'IPAD': '131.206.45.21',
    'LAST4': '2514',
    'MACK': 'Y',
    'MERC': '999666',
    'MODE': 'Q',
    'NAME': 'SdkTestFirstName SdkTestLastName',
    'PENC': 'KHASH',
    # 'PENC': '',
    'PROD_DESC[0]': '3000 CANDLEPOWER PLASMA FLASHLIGHT',
    'PROD_ITEM[0]': 'SG999999',
    'PROD_PRICE[0]': '68990',
    'PROD_QUANT[0]': '2',
    'PROD_TYPE[0]': 'SPORTING_GOODS',
    # 'PTOK': '0007380568572514',
    'PTOK': '000738F16NA2S935A5HY',  # for khashed=True in Payment
    'PTYP': 'CARD',
    'S2A1': '567 West S2A1 Court North',
    'S2A2': '',
    'S2CC': 'US',
    'S2CI': 'Gnome',
    'S2EM': 'sdkTestShipToEmail@kountsdktestdomain.com',
    'S2NM': 'SdkShipToFN SdkShipToLN',
    'S2PC': '99762',
    'S2PN': '555-777-1212',
    'S2ST': 'AK',
    'SDK': 'PYTH',
    'SDK_VERSION': 'Sdk-Ris-%s-%s' % (SDKConfig.LANG, SDKConfig.SDK_VERSION),
    'SITE': 'DEFAULT',
    'SPREMISE': '',
    'SSTREET': '',
    'TOTL': '123456',
    'VERS': SDKConfig.VERS,
}


def generate_unique_id():
    """unique session id"""
    return str(uuid.uuid4()).replace('-', '').upper()


def default_inquiry(merchant_id, session_id, email_client, payment):
    """default_inquiry, PENC is not set"""
    inq = Inquiry()
    inq.set_request_mode(InquiryMode.DEFAULT)
    inq.set_shipping_address(SHIPPING_ADDRESS)
    inq.set_shipping_name("SdkShipToFN SdkShipToLN")  # S2NM
    inq.set_billing_address(BILLING_ADDRESS)
```

(continues on next page)

(continued from previous page)

```
    inq.set_currency(CurrencyType.USD)  # CURR
    inq.set_total('123456')  # TOTL
    inq.set_billing_phone_number("555-867-5309")  # B2PN
    inq.set_shipping_phone_number("555-777-1212")  # S2PN
    inq.set_email_client(email_client)
    inq.set_customer_name("SdkTestFirstName SdkTestLastName")
    inq.set_unique_customer_id(session_id[:20])  # UNIQ
    inq.set_website("DEFAULT")  # SITE
    inq.set_email_shipping("sdkTestShipToEmail@kountsdktestdomain.com")
    inq.set_ip_address("4.127.51.215")  # IPAD
    cart_items = list()
    cart_items.append(CartItem("SPORTING_GOODS", "SG999999",
                               "3000 CANDLEPOWER PLASMA FLASHLIGHT",
                               '2', '68990'))
    inq.set_shopping_cart(cart_items)
    inq.version()
    inq.set_version(SDKConfig.VERS)  # 0695
    inq.set_merchant(merchant_id)
    inq.set_payment(payment)  # PTOK
    inq.set_session_id(session_id)  # SESS
    inq.set_order_number(session_id[:10])  # ORDR
    inq.set_authorization_status(AuthStatus.APPROVE)  # AUTH
    inq.set_avs_zip_reply(BankcardReply.MATCH)
    inq.set_avs_address_reply(BankcardReply.MATCH)
    inq.set_avs_cvv_reply(BankcardReply.MATCH)
    inq.set_merchant_acknowledgment(MerchantAcknowledgment.TRUE)  # "MACK"
    inq.set_cash('4444')
    return inq


@pytest.mark.usefixtures("api_url", "api_key", "merchant_id")
class TestInquiry(unittest.TestCase):
    """Inquiry class tests"""
    maxDiff = None

    def setUp(self):
        self.session_id = str(generate_unique_id())
        self.client = Client(self.api_url, self.api_key)

    def test_utilities(self):
        """test_utilities"""
        payment = Payment(
            payment_type="CARD",
            payment_token=PTOK,
            khashed=False)
        self.assertEqual(payment._payment_type, 'CARD')
        self.assertEqual(payment.last4, '2514')
        self.assertEqual(payment.payment_token, '0007380568572514')
        self.assertFalse(payment.khashed)
        inq = default_inquiry(
            merchant_id=self.merchant_id,
            session_id=self.session_id,
            email_client=EMAIL_CLIENT,
            payment=payment)

        expected_not_khashed = expected.copy()
        expected_not_khashed["PTOK"] = '0007380568572514'
```

(continues on next page)

(continued from previous page)

```
        actual = inq.params
        self.assertEqual(actual['PTYP'], 'CARD')
        self.assertIn(expected_not_khashed['SDK_VERSION'],
                      actual['SDK_VERSION'])

        del (actual['UNIQ'],
             actual['IPAD'],
             actual['SDK_VERSION'],
             actual['SESS'],
             actual['ORDR'],
             expected_not_khashed['SDK_VERSION'],
             expected_not_khashed['PENC'])

        self.assertEqual(actual, expected_not_khashed)

    def test_utilities_khashed(self):
        """test_utilities khashed"""
        _expected = expected.copy()
        payment = CardPayment(PTOK)
        self.assertEqual(payment._payment_type, 'CARD')
        self.assertEqual(payment.last4, '2514')
        self.assertEqual(payment.payment_token, '000738F16NA2S935A5HY')
        self.assertTrue(payment.khashed)
        result = default_inquiry(
            session_id=self.session_id,
            merchant_id=self.merchant_id,
            email_client=EMAIL_CLIENT,
            payment=payment)
        actual = result.params
        self.assertEqual(actual['PTYP'], 'CARD')
        self.assertIn(_expected['SDK_VERSION'], actual['SDK_VERSION'])
        del (actual['UNIQ'],
             actual['IPAD'],
             actual['SDK_VERSION'],
             actual['SESS'],
             actual['ORDR'],
             _expected['SDK_VERSION'])
        self.assertEqual(actual, _expected)

    def test_utilities_gift_khashed(self):
        """test_utilities GIFT khashed"""
        _expected = expected.copy()
        payment = GiftCardPayment(PTOK)
        self.assertEqual(payment._payment_type, 'GIFT')
        self.assertEqual(payment.last4, '2514')
        self.assertEqual(payment.payment_token, '000738F16NA2S935A5HY')
        self.assertTrue(payment.khashed)
        result = default_inquiry(
            session_id=self.session_id,
            merchant_id=self.merchant_id,
            email_client=EMAIL_CLIENT,
            payment=payment)
        actual = result.params
        self.assertEqual(actual['PTYP'], 'GIFT')
        self.assertIn(_expected['SDK_VERSION'], actual['SDK_VERSION'])
        del (_expected['SDK_VERSION'],
             _expected['PTYP'],
```

(continues on next page)

(continued from previous page)

```
            actual['PTYP'],
            actual['UNIQ'],
            actual['IPAD'],
            actual['SDK_VERSION'],
            actual['SESS'],
            actual['ORDR'])
        self.assertEqual(actual, _expected)


if __name__ == "__main__":
    unittest.main()
```

test_payment.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"Test Payment Type"
import unittest
import pytest

from kount.util.khash import Khash
from kount.util.payment import (
    BillMeLaterPayment, CardPayment, CheckPayment,
    GiftCardPayment, GooglePayment,
    GreenDotMoneyPakPayment, NoPayment,
    Payment, PaypalPayment)

from kount.version import VERSION
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


@pytest.mark.usefixtures("conf_key")
class TestPaymentType(unittest.TestCase):
    """Test Payment Type"""
    def setUp(self):
        self.test = 1234567890*1000000000

    def test_giftcardpayment(self):
        """giftcard payment"""
        ptype = GiftCardPayment(gift_card_number=self.test, khashed=False)
        self.assertTrue(isinstance(ptype, Payment))
        self.assertEqual(ptype.last4, str(self.test)[-4:])
        self.assertFalse(ptype.khashed)
        self.assertEqual(ptype.payment_type, "GIFT")
        self.assertEqual(ptype.payment_token, str(self.test))

    def test_payments(self):
        "all predefined payments"
```

(continues on next page)

(continued from previous page)

```
        plist = (Payment, BillMeLaterPayment, CardPayment,
                 CheckPayment, GiftCardPayment, GooglePayment,
                 GreenDotMoneyPakPayment, NoPayment, Payment, PaypalPayment)
        payment_dict = {
            "BLML": BillMeLaterPayment(self.test, khashed=False),
            "CARD": CardPayment(self.test, khashed=False),
            "CHEK": CheckPayment(self.test, khashed=False),
            "GIFT": GiftCardPayment(self.test, khashed=False),
            "GOOG": GooglePayment(self.test, khashed=False),
            "GDMP": GreenDotMoneyPakPayment(self.test, khashed=False),
            "NONE": NoPayment(),
            "PYPL": PaypalPayment(self.test, khashed=False),
            }
        ptypes = []
        for current in payment_dict:
            curp = payment_dict[current]
            if current == "NONE":
                self.assertEqual(curp.last4, "NONE")
                self.assertIsNone(curp.payment_token)
            else:
                self.assertEqual(curp.last4, str(self.test)[-4:])
                self.assertEqual(curp.payment_token, str(self.test))
            self.assertEqual(curp._payment_type, current)
            ptypes.append(payment_dict[current])
            self.assertIsInstance(payment_dict[current], plist)
            if curp.payment_token is not None:
                self.assertEqual(curp.payment_token, str(self.test))

    def test_user_defined_payment(self):
        "user defined payments"""
        curp = Payment("PM42", self.test, False)
        self.assertEqual(curp.last4, str(self.test)[-4:])
        self.assertEqual(curp.payment_token, str(self.test))
        self.assertFalse(curp.khashed)
        self.assertEqual(curp._payment_type, "PM42")
        self.assertEqual(curp.payment_token, str(self.test))
        self.assertIsInstance(curp, Payment)

    def test_user_defined_payment_khashed(self):
        "user defined payments with Payment - khashed token"
        curp = Payment("PM42", self.test, True)
        self.assertEqual(curp.last4, str(self.test)[-4:])
        self.assertEqual(curp.payment_token,
                         Khash.get().hash_payment_token(self.test))
        self.assertTrue(curp.khashed)
        self.assertEqual(curp._payment_type, "PM42")
        self.assertIsInstance(curp, Payment)

    def test_user_defined_newpayment(self):
        "user defined payments - token khashed and notkhashed "
        curp = Payment("PM42", self.test, khashed=False)
        self.assertEqual(curp.last4, str(self.test)[-4:])
        self.assertEqual(curp.payment_token, str((self.test)))
        self.assertFalse(curp.khashed)
        self.assertEqual(curp.payment_type, "PM42")
        self.assertIsInstance(curp, Payment)
        curp = Payment("PM42", self.test, True)
```

(continues on next page)

(continued from previous page)

```
        self.assertEqual(curp.last4, str(self.test)[-4:])
        self.assertEqual(curp.payment_token,
                         Khash.get().hash_payment_token(self.test))
        self.assertTrue(curp.khashed)


if __name__ == "__main__":
    unittest.main(
        #~ defaultTest="TestPaymentType.test_payments"
        )
```

test_validation_error.py

test_khash.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
import unittest
import pytest

from kount.version import VERSION
from kount.util.khash import Khash
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


@pytest.mark.usefixtures("conf_key")
class TestKhash(unittest.TestCase):
    """Khash class test cases"""

    def setUp(self):
        self.list_for_hash = ["4111111111111111",
                              '5199185454061655',
                              4259344583883]
        self.expected = ['WMS5YA6FUZA1KC', '2NOQRXNKTTFL11', 'FEXQI1QS6TH2O5']
        self.merchant_id = '666666'
        self.khash = Khash.get()

    def test_token_valid(self):
        """valid token"""
        self.assertEqual(
            "BADTOKGM3BD98ZY871QB",
            self.khash.hash_payment_token(token="BADTOKEN"))

        self.assertEqual(
            "000738F16NA2S935A5HY",
            self.khash.hash_payment_token(token="0007380568572514"))

        for i, plain_text in enumerate(self.list_for_hash):
```

(continues on next page)

(continued from previous page)

```
            card_hashed = self.khash.hash_payment_token(token=plain_text)
            expected = "%s%s" % (str(self.list_for_hash[i])[:6],
                                 self.expected[i])
            self.assertEqual(card_hashed, expected)
            self.assertTrue(self.khash.khashed(card_hashed))

    def test_token_invalid(self):
        """invalid token"""
        with self.assertRaises(ValueError):
            self.khash.hash_payment_token(token="")
        with self.assertRaises(ValueError):
            self.khash.hash_payment_token(token=None)
        card_hashed = self.khash.hash_payment_token(token="10**200")
        self.assertEqual(card_hashed, "10**20GA6AXR02LVUE5X")
        with self.assertRaises(ValueError):
            self.khash.hash_payment_token(token=-42)
        with self.assertRaises(ValueError):
            self.khash.hash_payment_token(token=10**200)
        with self.assertRaises(ValueError):
            self.khash.hash_payment_token(token=0)
        card_hashed = self.khash.hash_payment_token(token="Beatles")
        self.assertEqual(card_hashed, "Beatle5STRFTYPXBR14E")
        self.assertTrue(self.khash.khashed(card_hashed))
        bad = "John"
        try:
            self.khash.hash_payment_token(token=bad)
        except ValueError as vale:
            self.assertEqual("incorrect arg: [%s]" % bad, str(vale))
        with self.assertRaises(ValueError):
            self.assertTrue(self.khash.hash_payment_token(token=bad))

    def test_hash_gift_card(self):
        """gift card"""
        for i in range(len(self.list_for_hash)):
            card_hashed = self.khash.hash_gift_card(
                self.merchant_id, self.list_for_hash[i])
            expected = "%s%s" % (self.merchant_id, self.expected[i])
            self.assertEqual(card_hashed, expected)
            self.assertTrue(self.khash.khashed(card_hashed))

    def test_hash_gift_card_int_merchantid(self):
        """test_hash_gift_card_int_merchantid"""
        for i in range(len(self.list_for_hash)):
            card_hashed = self.khash.hash_gift_card(
                self.merchant_id, self.list_for_hash[i])
            expected = "%s%s" % (self.merchant_id, self.expected[i])
            self.assertEqual(card_hashed, expected)
            self.assertTrue(self.khash.khashed(card_hashed))

    def test_list_for_hash_empty(self):
        """list_for_hash_empty"""
        list_for_hash = ""
        with self.assertRaises(ValueError):
            self.khash.hash_gift_card(self.merchant_id, list_for_hash)

    def test_list_for_hash_none(self):
        """hash_none"""
```

(continues on next page)

(continued from previous page)

```
        list_for_hash = None
        with self.assertRaises(ValueError):
            self.khash.hash_gift_card(self.merchant_id, list_for_hash)

    def test_gift_card_empty_values(self):
        """gift_card_empty_values"""
        list_for_hash = []
        with self.assertRaises(ValueError):
            self.khash.hash_gift_card(self.merchant_id, list_for_hash)

    def test_gift_card_no_merchant(self):
        """gift card without merchant"""
        list_for_hash = []
        merchant_id = ""
        with self.assertRaises(ValueError):
            self.khash.hash_gift_card(merchant_id, list_for_hash)

    def test_gift_card_merchant_empty_str(self):
        """gift_card_merchant_empty_str"""
        merchant_id = ""
        with self.assertRaises(ValueError):
            self.khash.hash_gift_card(merchant_id, self.list_for_hash)

    def test_list_for_hash_merchant_none(self):
        """list_for_hash_merchant_none"""
        list_for_hash = []
        merchant_id = None
        with self.assertRaises(ValueError):
            self.khash.hash_gift_card(merchant_id, list_for_hash)

    def test_list_for_hash_args_missing(self):
        """list_for_hash_args_missing"""
        list_for_hash = None
        merchant_id = None
        with self.assertRaises(ValueError):
            self.khash.hash_gift_card(merchant_id, list_for_hash)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

test_ris_test_suite.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"""Test Cases from sdk documentation"""
import unittest

import pytest

from kount.request import (AuthStatus, BankcardReply, InquiryMode,
                           CurrencyType, MerchantAcknowledgment)
from kount.request import Update, UpdateMode
from kount.util.khash import Khash
```

(continues on next page)

(continued from previous page)

```
from kount.client import Client
from kount.util.cartitem import CartItem
from kount.util.payment import CardPayment
from kount.version import VERSION

from .test_basic_connectivity import generate_unique_id, default_inquiry

from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS

# raise_errors - if  True - raise errors instead of logging in debugger
_RAISE_ERRORS = False

PTOK = "0007380568572514"
EMAIL_CLIENT = "sdkTest@kountsdktestdomain.com"


@pytest.mark.usefixtures("merchant_id", "api_key", "api_url")
class TestRisTestSuite(unittest.TestCase):
    """Ris Test Suite
        default logging errors instead fo raising
        to raise errors - put raise_errors=True in Client:
        Client(url=URL_API, key=KOUNT_API_KEY,
               timeout=TIMEOUT, RAISE_ERRORS=True)
    """

    maxDiff = None

    def setUp(self):
        self.session_id = generate_unique_id()[:32]
        self.payment = CardPayment(PTOK, khashed=False)
        self.client = Client(self.api_url, self.api_key,
                             raise_errors=_RAISE_ERRORS)

    def inquiry(self):
        return default_inquiry(
            merchant_id=self.merchant_id,
            session_id=self.session_id,
            email_client=EMAIL_CLIENT,
            payment=self.payment)

    def test_1_ris_q_1_item_required_field_1_rule_review(self):
        """test_1_ris_q_1_item_required_field_1_rule_review"""
        res = self.client.process(self.inquiry())
        self.assertIsNotNone(res)
        self.assertEqual("R", res.get_auto())
        self.assertEqual(0, len(res.get_warnings()))
        expected = ['Review if order total > $1000 USD']
        actual = sorted(res.get_rules_triggered().values())
        self.assertEqual(expected, actual)
        self.assertEqual(self.session_id, res.get_session_id())
        self.assertEqual(res.get_session_id()[:10], res.get_order_id())
```

(continues on next page)

(continued from previous page)

```
    def test_2_ris_q_multi_cart_items2optional_fields2rules_decline(self):
        """test_2_ris_q_multi_cart_items2optional_fields2rules_decline
        cart_item - PROD_TYPE[0, PROD_ITEM[0], PROD_DESC[0]
                    PROD_QUANT[0],PROD_PRICE[0]"""
        inq = self.inquiry()
        inq.set_user_agent(
            "Mozilla/5.0 (Macintosh; "
            "Intel Mac OS X 10_9_5) AppleWebKit/537.36 "
            "(KHTML, like Gecko) Chrome/37.0.2062.124 "
            "Safari/537.36")
        inq.set_total(123456789)
        cart_items = [
            CartItem(
                "cart item type 0", "cart item 0",
                "cart item 0 description", 10, 1000),
            CartItem(
                "cart item type 1", "cart item 1",
                "cart item 1 description", 11, 1001),
            CartItem(
                "cart item type 2", "cart item 2",
                "cart item 1 description", 12, 1002)]
        inq.set_shopping_cart(cart_items)
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual("D", res.get_auto())
        self.assertEqual(0, len(res.get_warnings()))
        expected = sorted(
            {'1024842': 'Review if order total > $1000 USD',
             '1024844': 'Decline if order total > $1000000 USD'}.values())
        actual = sorted(res.get_rules_triggered().values())
        self.assertEqual(expected, actual)

    def test_3_ris_q_with_user_defined_fields(self):
        """test_3_ris_q_with_user_defined_fields"""
        udf1 = "ARBITRARY_ALPHANUM_UDF"
        udf2 = "ARBITRARY_NUMERIC_UDF"
        inq = self.inquiry()
        inq.set_user_defined_field(udf1, "alphanumeric trigger value")
        inq.set_user_defined_field(udf2, "777")
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual("R", res.get_auto())
        self.assertEqual(3, len(res.get_rules_triggered()))
        self.assertEqual(0, len(res.get_warnings()))
        self.assertEqual(0, len(res.get_errors()))
        self.assertEqual(0, len(res.get_counters_triggered()))
        expected = sorted(
            {'1025086': 'review if %s contains "trigger"' % udf1,
             '1024842': 'Review if order total > $1000 USD',
             '1025088': "review if %s == 777" % udf2}.values())
        actual = sorted(res.get_rules_triggered().values())
        self.assertEqual(expected, actual)

    def test_4_ris_q_hard_error_expected(self):
        """test_4_ris_q hard_error_expected,
        overwrite the PTOK value to induce an error in the RIS"""
```

(continues on next page)

(continued from previous page)

```
        inq = self.inquiry()
        inq.params["PENC"] = "KHASH"
        inq.params["PTOK"] = "BADPTOK"
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual(
            ["332 BAD_CARD Cause: [PTOK invalid format], "
             "Field: [PTOK], Value: [hidden]"],
            res.get_errors())
        self.assertEqual("E", res.get_mode())
        self.assertEqual(332, res.get_error_code())
        self.assertEqual(0, len(res.get_warnings()))

    def test_5_ris_q_warning_approved(self):
        """test_5_ris_q_warning_approved"""
        inq = self.inquiry()
        inq.set_total(1000)
        label = "UDF_DOESNOTEXIST"
        mesg = "throw a warning please!"
        inq.set_user_defined_field(label, mesg)
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual("A", res.get_auto())
        self.assertEqual(2, len(res.get_warnings()))
        self.assertEqual(res.get_warnings()[0],
                         "399 BAD_OPTN Field: [UDF], Value: "
                         "[%s=>%s]" % (label, mesg))
        self.assertEqual(res.get_warnings()[1],
                         "399 BAD_OPTN Field: [UDF], Value: "
                         "[The label [%s]"
                         " is not defined for merchant ID [%s].]" % (
                             label, self.merchant_id))

    def test_6_ris_q_hard_soft_errors_expected(self):
        """test_6_ris_q_hard_soft_errors_expected"""
        inq = self.inquiry()
        inq.params["PENC"] = "KHASH"
        inq.params["PTOK"] = "BADPTOK"
        label = "UDF_DOESNOTEXIST"
        mess = "throw a warning please!"
        inq.params["UDF[%s]" % label] = mess
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual("E", res.get_mode())
        self.assertEqual(332, res.get_error_code())
        self.assertEqual(1, len(res.get_errors()))
        self.assertEqual(
            [("332 BAD_CARD Cause: [PTOK invalid format], "
              "Field: [PTOK], Value: [hidden]")],
            res.get_errors())
        warnings = res.get_warnings()
        self.assertEqual(2, len(warnings))
        self.assertEqual(
            "399 BAD_OPTN Field: [UDF], Value: [%s=>%s]"
            % (label, mess), warnings[0])
        self.assertEqual(
            "399 BAD_OPTN Field: [UDF], Value: [The label [%s] "
```

(continues on next page)

(continued from previous page)

```
            "is not defined for merchant ID [%s].]"
            % (label, self.merchant_id), warnings[1])

    def test_7_ris_w2_kc_rules_review(self):
        """test_7_ris_w2_kc_rules_review"""
        inq = self.inquiry()
        inq.set_request_mode(InquiryMode.WITH_THRESHOLDS)
        inq.set_total(10001)
        inq.set_kount_central_customer_id("KCentralCustomerOne")
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual(res.get_kc_decision(), 'R')
        self.assertEqual(len(res.get_kc_warnings()), 0)
        self.assertEqual(len(res.get_kc_events()), 2)
        events = res.get_kc_events()
        print(events)
        self.assertEqual(events[0].code, 'billingToShippingAddressReview')
        self.assertEqual(events[1].expression, '10001 > 10000')
        self.assertEqual(events[0].decision, 'R')
        self.assertEqual(events[1].code, 'orderTotalReview')
        self.assertEqual(events[0].expression, '5053 > 1')
        self.assertEqual(events[1].decision, 'R')

    def test_8_ris_j_1_kount_central_rule_decline(self):
        """test_8_ris_j_1_kount_central_rule_decline"""
        inq = self.inquiry()
        inq.set_request_mode(InquiryMode.JUST_THRESHOLDS)
        inq.set_total(1000)
        inq.set_kount_central_customer_id("KCentralCustomerDeclineMe")
        if not _RAISE_ERRORS:
            res = self.client.process(inq)
            self.assertIsNotNone(res)
            self.assertEqual("D", res.get_kc_decision())
            self.assertEqual(0, len(res.get_kc_warnings()))
            kc_events = res.get_kc_events()
            self.assertEqual(1, len(kc_events), )
            self.assertEqual(kc_events[0].code, "orderTotalDecline")

    def test_9_mode_u_after_mode_q(self):
        """test_9_mode_u_after_mode_q"""
        res = self.client.process(self.inquiry())
        self.assertIsNotNone(res)
        transaction_id = res.get_transaction_id()
        session_id = res.get_session_id()
        order_id = res.get_order_id()

        update1 = Update()
        update1.set_mode(UpdateMode.NO_RESPONSE)
        update1.set_transaction_id(transaction_id)
        update1.set_merchant(self.merchant_id)
        update1.set_session_id(session_id)
        update1.set_order_number(order_id)
        # PTOK has to be khashed manually because of its explicit setting
        token_new = "5386460135176807"
        update1.params["PTOK"] = Khash.get().hash_payment_token(token_new)
        update1.params["LAST4"] = token_new[-4:]
        update1.params["FRMT"] = 'JSON'
```

(continues on next page)

(continued from previous page)

```
        update1.set_khash_payment_encoding(True)
        update1.set_merchant_acknowledgment(MerchantAcknowledgment.TRUE)
        update1.set_authorization_status(AuthStatus.APPROVE)
        update1.set_avs_zip_reply(BankcardReply.MATCH)
        update1.set_avs_address_reply(BankcardReply.MATCH)
        update1.set_avs_cvv_reply(BankcardReply.MATCH)
        res = self.client.process(update1)
        self.assertIsNotNone(res)
        self.assertEqual("U", res.get_mode())
        self.assertIsNone(res.get_geox())
        self.assertIsNone(res.get_score())
        self.assertIsNone(res.get_auto())

    def test_10_mode_x_after_mode_q(self):
        """test_10_mode_x_after_mode_q
        PTOK has to be khashed manually because of
        its explicit setting"""
        res = self.client.process(self.inquiry())
        self.assertIsNotNone(res)
        transaction_id = res.get_transaction_id()
        session_id = res.get_session_id()
        order_id = res.get_order_id()
        update1 = Update()
        update1.set_mode(UpdateMode.WITH_RESPONSE)
        update1.set_transaction_id(transaction_id)
        update1.set_merchant(self.merchant_id)
        update1.set_session_id(session_id)
        update1.set_order_number(order_id)
        token_new = "5386460135176807"
        update1.set_khash_payment_encoding(self.payment.khashed)
        if self.payment.khashed:
            token_new = Khash.get().hash_payment_token(token_new)
        update1.params["PTOK"] = token_new
        update1.params["LAST4"] = token_new[-4:]
        update1.params["FRMT"] = 'JSON'
        update1.set_merchant_acknowledgment(MerchantAcknowledgment.TRUE)
        update1.set_authorization_status(AuthStatus.APPROVE)
        update1.set_avs_zip_reply(BankcardReply.MATCH)
        update1.set_avs_address_reply(BankcardReply.MATCH)
        update1.set_avs_cvv_reply(BankcardReply.MATCH)
        res = self.client.process(update1)
        self.assertIsNotNone(res)
        self.assertEqual("X", res.get_mode())
        self.assertIsNotNone(res.get_geox())
        self.assertIsNotNone(res.get_score())
        self.assertIsNotNone(res.get_auto())

    def test_11_mode_p(self):
        res = self.client.process(self.inquiry())
        self.assertIsNotNone(res)
        inq = self.inquiry()
        inq.set_request_mode(InquiryMode.PHONE)
        inq.set_anid("2085551212")
        inq.set_total(1000)
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual("P", res.get_mode())
```

(continues on next page)

(continued from previous page)

```
        self.assertEqual("A", res.get_auto())

    def test_14_ris_q_using_payment_encoding_mask_valid(self):
        """test_14_ris_q_using_payment_encoding_mask_valid"""
        ptok_2 = "370070XXXXX9797"
        last4 = ptok_2[-4:]
        penc = 'MASK'
        res = self.client.process(self.inquiry())
        self.assertIsNotNone(res)
        inq = self.inquiry()
        inq.params['LAST4'] = last4
        inq.params['PTOK'] = ptok_2
        inq.params['PENC'] = penc
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual("AMEX", res.get_brand())

    def test_15_ris_q_using_payment_encoding_mask_error(self):
        """test_15_ris_q_using_payment_encoding_mask_error"""
        ptok_2 = "370070538959797"
        last4 = ptok_2[-4:]
        penc = 'MASK'
        inq = self.inquiry()
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        inq.params['LAST4'] = last4
        inq.params['PTOK'] = ptok_2
        inq.params['PENC'] = penc
        res = self.client.process(inq)
        self.assertIsNotNone(res)
        self.assertEqual({
            'ERRO': 340,
            'ERROR_0':
                '340 BAD_MASK Cause: [value [%s] did not match regex '
                '/^\\d{6}X{5,9}\\d{1,4}$/], Field: [PTOK], Value: '
                '[%s]' % (ptok_2, ptok_2),
            'ERROR_COUNT': 1,
            'MODE': 'E',
            'WARNING_COUNT': 0}, res.params)


class TestRisTestSuiteKhashed(TestRisTestSuite):
    """Ris Test Suite Khashed
        default logging errors instead fo raising
        to raise errors - put raise_errors=True in Client:
        Client(url=URL_API, key=KOUNT_API_KEY,
               timeout=TIMEOUT, RAISE_ERRORS=True)
    """
    maxDiff = None

    def setUp(self):
        self.session_id = generate_unique_id()[:32]
        self.payment = CardPayment(PTOK)
        self.client = Client(self.api_url, self.api_key,
                             raise_errors=_RAISE_ERRORS)
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    unittest.main(verbosity=2)
```

test_ris_validator.py

test_xmlparser.py

test_bed_examples.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
"""Test Cases for an example implementation
generate_unique_id
put test data in user_inquiry
"""
import unittest
import pytest

from kount.client import Client
from kount.config import SDKConfig
from kount.util.payment import CardPayment
from kount.inquiry import Inquiry
from kount.request import (AuthStatus, BankcardReply, InquiryMode,
                           CurrencyType, MerchantAcknowledgment)
from kount.util.cartitem import CartItem
from kount.util.address import Address
from kount.version import VERSION

from .test_inquiry import generate_unique_id

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


PTOK = "4111111111111111"
EMAIL = 'john@test.com'
BILLING_ADDRESS = Address("", "", "Manchester", "NH", "03109", "US")
BILLING_PHONE = "555-888-5678"


def user_inquiry(session_id, merchant_id, email_client, payment):
    """user_inquiry, PENC is not set"""
    result = Inquiry()
    result.set_request_mode(InquiryMode.DEFAULT)
    result.set_billing_address(BILLING_ADDRESS)
    result.set_currency(CurrencyType.USD)  # CURR
    result.set_total(3500)  # TOTL
    result.set_billing_phone_number(BILLING_PHONE)  # B2PN
    result.set_email_client(email_client)
    result.set_customer_name("J Test")
    result.set_unique_customer_id(session_id[:20])  # UNIQ
```

(continues on next page)

(continued from previous page)

```
    result.set_website("DEFAULT")  # SITE
    # result.set_ip_address("4.127.51.215")  # IPAD
    result.set_ip_address('2001:0:3238:DFE1:63::FEFB')  # IPAD
    cart_items = [CartItem("1", "8482", "Standard Monthly Plan", 1, '3500')]
    result.set_shopping_cart(cart_items)
    result.version()
    result.set_version(SDKConfig.VERS)  # 0710
    result.set_merchant(merchant_id)
    result.set_payment(payment)  # PTOK
    result.set_session_id(session_id)  # SESS
    result.set_order_number(session_id[:10])  # ORDR
    result.set_authorization_status(AuthStatus.APPROVE)  # AUTH
    result.set_avs_zip_reply(BankcardReply.MATCH)
    result.set_avs_address_reply(BankcardReply.MATCH)
    result.set_avs_cvv_reply(BankcardReply.MATCH)
    result.set_merchant_acknowledgment(MerchantAcknowledgment.TRUE)  # "MACK"
    return result


expected = {
    'ANID': '',
    'AUTH': 'A',
    'AVST': 'M',
    'AVSZ': 'M',
    'B2A1': '',
    'B2A2': '',
    'B2CC': 'US',
    'B2CI': 'Manchester',
    'B2PC': '03109',
    'B2PN': BILLING_PHONE,
    'B2ST': 'NH',
    'BPREMISE': '',
    'BSTREET': '',
    'CURR': 'USD',
    'CVVR': 'M',
    'EMAL': EMAIL,
    'FRMT': 'JSON',
    'IPAD': '2001:0:3238:DFE1:63::FEFB',
    'LAST4': '1111',
    'MACK': 'Y',
    'MERC': '999666',
    'MODE': 'Q',
    'NAME': 'J Test',
    # 'ORDR': '4F7132C2FE',
    # 'PENC': 'KHASH',
    'PROD_DESC[0]': 'Standard Monthly Plan',
    'PROD_ITEM[0]': '8482',
    'PROD_PRICE[0]': '3500',
    'PROD_QUANT[0]': 1,
    'PROD_TYPE[0]': '1',
    'PTOK': PTOK,
    'PTYP': 'CARD',
    'SDK': 'PYTH',
    # 'SDK_VERSION': 'Sdk-Ris-Python-0695-201708301601',
    # 'SESS': '4F7132C2FE8547928CD9329B78AA0A59',
    'SITE': 'DEFAULT',
    'TOTL': 3500,
```

(continues on next page)

(continued from previous page)

```
    # 'UNIQ': '4F7132C2FE8547928CD9',
    'VERS': '0720'}


@pytest.mark.usefixtures("api_url", "api_key", "merchant_id")
class TestBed(unittest.TestCase):
    """Test Bed for use-cases, with & without Khash"""
    maxDiff = None

    def setUp(self):
        self.session_id = generate_unique_id()[:32]
        self.email_client = EMAIL

    def test_not_khashed(self):
        """test without khashed card"""
        # required khashed=False
        payment = CardPayment(PTOK, False)
        self.inq = user_inquiry(
            self.session_id, self.merchant_id, self.email_client,
            payment=payment)
        self.assertNotIn('PENC', self.inq.params)
        self.compare(expected)

    def test_khashed(self):
        """test with khashed card"""
        # not required default khashed=True
        payment = CardPayment(PTOK)
        self.inq = user_inquiry(
            self.session_id, self.merchant_id, self.email_client,
            payment=payment)
        self.assertIn('PENC', self.inq.params)
        self.assertEqual('KHASH', self.inq.params['PENC'])
        expected_khashed = expected.copy()
        expected_khashed['PENC'] = 'KHASH'
        expected_khashed['PTOK'] = '411111WMS5YA6FUZA1KC'
        self.compare(expected_khashed)

    def compare(self, expected_dict):
        """common method for both tests"""
        res = Client(self.api_url, self.api_key).process(self.inq)
        self.assertIsNotNone(res)
        self.assertNotIn('ERRO', repr(res))
        actual = self.inq.params.copy()
        remove = ['SDK_VERSION', 'SESS', 'UNIQ', 'ORDR']
        for k in remove:
            if k in actual:
                del actual[k]
        self.assertEqual(expected_dict, actual)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

test_base85_encode_decode.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

(continues on next page)

(continued from previous page)

```
# This file is part of the Kount python sdk project
# https://github.com/Kount/kount-ris-python-sdk/)
# Copyright (C) 2017 Kount Inc. All Rights Reserved.
import unittest

import sys
from kount.util.a85 import a85decode, a85encode
from kount.version import VERSION
from kount.config import SDKConfig

__author__ = SDKConfig.SDK_AUTHOR
__version__ = VERSION
__maintainer__ = SDKConfig.SDK_MAINTAINER
__email__ = SDKConfig.MAINTAINER_EMAIL
__status__ = SDKConfig.STATUS


class Base85EncodeDecodeTest(unittest.TestCase):
    """Base85EncodeDecodeTest"""

    plain_text = "This is sample text for testing purposes."
    encoded_text = b"<+oue+DGm>F(&p)Ch4`2AU&;>AoD]4FCfN8Bl7Q+E-62?Df]K2/c"

    def test_encode(self):
        """test valid encode"""
        encoded = a85encode(self.plain_text.encode('utf-8'))
        decoded = a85decode(encoded)
        self.assertEqual(encoded, self.encoded_text)
        self.assertEqual(decoded, self.plain_text.encode('utf-8'))

    def test_decode(self):
        """test valid decode"""
        decoded = a85decode(self.encoded_text)
        self.assertEqual(decoded, self.plain_text.encode('utf-8'))

    def test_decode_invalid(self):
        """test invalid decode"""
        self.assertEqual(a85decode(b''), b'')
        self.assertRaises(ValueError, a85decode, self.plain_text)

    def test_encode_invalid(self):
        """test invalid encode"""
        self.assertEqual(a85encode(b''), b'')
        if sys.version_info[0] > 2:  # TODO
            self.assertRaises(TypeError, a85encode, '')
            self.assertRaises(TypeError, a85encode, self.plain_text)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

CHAPTER 39

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

K