Solitude Documentation

Release 0.1

Mozilla

February 05, 2016

Contents

Please note: this project is currently unmaintained and is not (or soon will not) be in active use by Mozilla.

Solitude is a payments server for processing payments for Mozilla's Marketplace and Addons site.

It provides a REST API for processing payments that you would plug into your site. We've implemented the APIs that we want to use for the Marketplace, not every API that the provider supports.

Currently we support:

- some Bango APIs
- some Braintree APIs
- some Zippy compliance

In the past PayPal was supported, that has been removed.

This project is based on **playdoh**. Mozilla's Playdoh is an open source web application template based on Django.

This document is available as a PDF.

Solitude is also a nice tasting beer from Brewery Vivant. The logo is theirs.

CHAPTER 1

Contents

1.1 Setup

The recommended way to run solitude is in Docker.

For running solitude in the marketplace environment, we recommend using Docker and reading the marketplace docs.

For running solitude in the Payments for Firefox Accounts, we recommend using Docker and reading the payments docs.

1.1.1 Requirements

If you don't have Python or MySQL installed. On OS X, homebrew is recommended:

```
brew install python mysql
```

Don't forget to set the default mysql password for your *root* user in that case (an empty password is possible):

```
mysql -uroot -p
```

Optionally install a virtualenv.

1.1.2 Install

From github:

```
git clone git://github.com/mozilla/solitude.git
```

If you used a virtualenv activate it and compile some playdoh dependencies:

```
cd solitude
pip install --no-deps -r requirements/dev.txt
```

1.1.3 Configure

Solitude will work without any settings changes at all with zippy, the reference implementation of the payment provider.

However since solitudes main job is to communicate with remote payment providers you will need to configure those. To do so create a settings file.

Create an empty settings:

```
cd solitude/settings
echo "from . import base" > local.py
```

1.1.4 Running tests

To run unit tests:

```
python manage.py test
```

All live server and Braintree integration tests are not run by default. To run live server and Braintree integration tests:

```
LIVE_TESTS=live,braintree python manage.py test
```

The value of LIVE_TESTS is passed to the nose args command.

For the Braintree tests to pass, you will need to have setup a Braintree sandbox account.

Bango settings

To process payments with Bango, you will need a Bango account. Once you have that, setup your account details:

```
BANGO_AUTH = {'USER': 'the.bango.username',
              'PASSWORD': 'the.bango.password'}
```

Solitude also receives requests from Bango. Bango needs to know a URL and a username and password for them. Example:

```
BANGO_BASIC_AUTH = {'USER': 'a.username',
                    'PASSWORD': 'a.password'}
BANGO_NOTIFICATION_URL = 'https://your.site/notification'
```

These are passed to Bango each time a package is created.

You can fake out Bango for some tasks if you'd like:

```
BANGO_MOCK = True
```

Braintree settings

To process payments for Braintree, you will need to get a Braintree account. For development, use a Braintree sandbox account.

Then go to Account > My User > API Keys.

These values can be set as environment variables (recommended if you're running in Docker):

```
export BRAINTREE_MERCHANT_ID=your-merchant-id
export BRAINTREE_PUBLIC_KEY=your-public-key
export BRAINTREE_PRIVATE_KEY=your-private-key
```

Alternatively you can add them to your local python configuration file:

```
BRAINTREE_MERCHANT_ID = 'your-merchant-id'
BRAINTREE_PUBLIC_KEY = 'your-public-key'
BRAINTREE_PRIVATE_KEY = 'your-private-key'
```

The Braintree API server is configured by this setting:

```
BRAINTREE_ENVIRONMENT = 'sandbox'
```

Zippy settings

Solitude supports zippy by default. If you'd like to use a server other than paas, then alter *ZIPPY_CONFIGURATION*, for example:

```
ZIPPY_CONFIGURATION = {
    'reference': {
        'url': 'http://localhost:8080',
        'auth': {'key': 'a.key',
                 'secret': 'a.secret',
                 'realm': 'a.realm'}
    }
}
```

- *reference*: this is the name of the zippy implementation. Its used as the key for the URLs.
- *url*: the location of the zippy server.
- *auth*: the key, secret and realm used for calculating the oAuth. Zippy must have the same configuration.

1.1.5 Optional settings

- **DUMP_REQUESTS**: *True* or *False*. Will dump to the *s.dump* log: incoming requests, outgoing requests and incoming responses.
- **CLEANSED_SETTINGS_ACCESS**: *True* or *False*. Will give you access to the cleansed settings in the *django.conf.settings* through the API. Should be *False* on production.

1.2 Security

1.2.1 Encryption

Currently we use django-aesfield to provide encryption on key fields. We'd recommend more levels of database encryption or file system encryption.

The encryption uses AES to do this.

Encrypted fields:

- buyers email
- sellers secret
- bango signature

The keys per field are mapped in settings. See *Setup* for more.

1.2.2 Hashed fields

Fields are

- buyers pin

- buyers new pin

1.2.3 Requests

All requests use OAuth 1.1 which signs the header using a secret key. Requests must be signed with that key or be rejected.

1.3 Overall

There are things that are assumed to be true on all API requests to solitude. And to prevent repetition in the documentation are mentioned here.

1.3.1 Requests

- It's assumed that all requests are *application/json*. This isn't enforced yet, but could be. If you use curling, all requests are sent with *Content-Type: application/json*.
- All requests should include *Accept: application/json*. If you use curling this is the case.

1.3.2 Responses

- It's assumed that all responses are *application/json*.
- Where possible we use standard HTTP error status codes, e.g 404. If there is anything unusual in the responses, we document those in the specific API calls.

Common elements in some responses:

- *response_pk* (string): a primary key for that resource. Will be unique to that resource. Example: *123*.
- *response_uri* (string): a URI to the object within solitude. To turn it into a URL add the protocol and domain of the server. Example: */generic/transaction/123/*. To turn that into URL: *http://solitude:2602/generic/transaction/123/*.
- *created* (datetime): when the object is created. Using the Django Rest Framework format, ECMA 262.
- *counter* (int): a counter that increments on each save of an object. This is used for etag generation.
- *modified* (datetime): when the object was last modified. Using the Django Rest Framework format, ECMA 262.

1.3.3 Errors

Errors. Consistent interface in progress and tracked by this issue.

To seperate the old and new style, two different kinds of errors will be returned a status of 400 for old format errors and 422 for new format errors.

400

Responses are currently inconsistent and pending upon fixes to Bango and upgrading to Django Rest Framework 3.x.

422

Errors will be raised with the namespace of the error, currently one of *mozilla*, *braintree* or *bango* to represent the part of the system that caused the error.

1.3.4 Mozilla

Form errors in Solitude are given the namespace *mozilla*.

An error contains the field the error occurred on and the message and code. It is possible for more than one error to exist on a field. For a consistent interface use the *code* attribute.

Example failure in form processing:

```
.. code:json::

    {
        "mozilla": {
            "name": [
                {"message": "First error", "code": "first"},
                {"message": "Second error", "code": "second"}
            ],
            "__all__": [
                {"message": "Non field error", "code": "non-field"}
            ]
        }
    }
```

In this example *name* is a field passed in the request. The *__all__* refers to an error that did not exist on a field.

1.3.5 Braintree

Data errors in Braintree are given the namespace *braintree*.

An error contains the field the error occurred on and the message and code. It is possible for more than one error to exist on a field. For a consistent interface use the code attribute, the *code* attribute is referenced in the Braintree documentation

Errors occur on Braintree fields, not fields passed in the request, so the the error keys do not match request fields.

Braintree has some errors which it doesn't consider validation errors because they are not specific to a submitted input field. However, Solitude still displays these as validation errors so that error handling is consistent.

Example failure from Braintree:

```
.. code:json::

    {
        "braintree": {
            "payment_method_token": [
                {"message": "Payment method token is invalid.", "code": "91903"}
            ],
            "__all__": [
                {"message": "Credit card denied", "code": "2000"}
            ]
        }
    }
```

If no error can be found solitude will add the message to the response with code of *unknown*, for example:

```
.. code:json::

    {
        "braintree": {
            "__all__": [
                {"message": "Invalid Secure Payment Data", "code": "unknown"}
            ]
        }
    }
```

1.4 Authentication

Most API requests can enforce zero-legged OAuth by having a shared key and secret on the servers. This allows solitude to check the client sending requests is allowed to do so. By default, this is *True*:

```
REQUIRE_OAUTH = True
CLIENT_OAUTH_KEYS = {
    'marketplace': 'please change this',
    'webpay': 'please change this',
}
```

In development, you might want to connect with curl and other tools. For that alter the *REQUIRE_OAUTH* setting to *False*.

Note: Service URLs do not require JWT encoding.

1.5 Generic

The generic API can be used for buyers and sellers.

1.5.1 Buyers

Buyers are identified by a UUID, which is a string (max 255 chars) that makes sense to the client. It must be unique within solitude, so we'd recommend prefixing the UUID, eg: `marketplace:<your-uuid>`

Note: after spartacus was merged some of these fields and features were no longer needed but remain in service.

Create

Buyers are added to solitude by an HTTP *POST* call. The POST should contain a unique UUID as well as the PIN the buyer has chosen:

POST /generic/buyer/

Request JSON Object

- **uuid** (*string*) – a unique identifier string.
- **active** (*bool*) – True if this is an active account.

- **authenticated** (*bool*) – True if this user account was authenticated in a trusted way. For example, if the email has been verified using the Firefox Accounts OAuth API then you would say this account has been authenticated.
- **email** (*string*) – email address.
- **locale** (*string*) – ISO 639 locale code to indicate the buyer's preferred locale. See the example below for how it looks.

```
{
    "uuid": "93e33277-87f7-417b-8ed2-371672b5297e",
    "email": "someone@somewhere.org",
    "locale": "fr,en;q=0.7,en-US;q=0.3"
}
```

Response

```
{
    "active": false,
    "counter": null,
    "email": "someone@somewhere.org",
    "locale": "fr,en;q=0.7,en-US;q=0.3",
    "needs_pin_reset": false,
    "new_pin": false,
    "pin": false,
    "pin_confirmed": false,
    "pin_failures": 0,
    "pin_is_locked_out": false,
    "pin_was_locked_out": false,
    "resource_pk": 4,
    "resource_uri": "/generic/buyer/4/",
    "uuid": "93e33277-87f7-417b-8ed2-371672b5297e"
}
```

Status Codes

- 201 Created – successfully processed.

List and retrieve

List buyers:

GET /generic/buyer/

You can filter on the following parameters.

Parameters

- **active** – the active flag for a user.
- **email** – users email address.
- **uuid** – the uuid for a user.

Response

A standard listing response containing buyers (see below).

Get the details of a buyer:

GET /generic/buyer/int:id/

Response

```
{
    "active": false,
    "counter": null,
    "email": "",
    "locale": "fr,en;q=0.7,en-US;q=0.3",
    "needs_pin_reset": false,
    "new_pin": false,
    "pin": false,
    "pin_confirmed": false,
    "pin_failures": 0,
    "pin_is_locked_out": false,
    "pin_was_locked_out": false,
    "resource_pk": 4,
    "resource_uri": "/generic/buyer/4/",
    "uuid": "93e33277-87f7-417b-8ed2-371672b5297e"
}
```

Parameters

- **email** – users email address.
- **type** – string
- **locale** – users locale, most likely the Accept Language HTTP header
- **type** – string
- **pin** – in a POST a PIN is a string, but in responses, the PIN is never returned. It returns a boolean, *true* if a PIN is present, *false* if not.
- **type** – boolean
- **pin_confirmed** – if the pin has been confirmed.
- **type** – boolean
- **new_pin** – a *new_pin* so that a confirmation can be made.
- **type** – boolean
- **active** – if the buyer is currently active or not, defaults to *true*
- **type** – boolean
- **pin_locked_out** – if the PIN is currently locked out.
- **type** – boolean
- **pin_failures** – the number of failed PIN entries. Reset to 0 on successful entry. When a threshold is reached defined by *PIN_FAILURES* in settings.
- **type** – int

Confirm PIN

Once you have created a buyer with a PIN, you'll need to have the buyer confirm their PIN. Once you've received their confirmed PIN you can POST to the `confirm_pin` endpoint like so:

POST /generic/confirm_pin/

Request

```
{
    "uuid": "93e33277-87f7-417b-8ed2-371672b5297e",
    "pin": "8472"
}
```

Response

```
{
    "confirmed": false,
    "uuid": "93e33277-87f7-417b-8ed2-371672b5297e"
}
```

Status Codes

- 200 OK – uuid found and PIN processed, check *confirmed* in the result
- 404 Not Found – uuid not found.

Parameters

- **confirmed** (*boolean*) – if *true* the PIN matched, if *false* the PIN did not match.

Verify PIN

Once you have a buyer with a confirmed pin, the next time they go to purchase something you can simply verify their PIN using the `verify_pin` endpoint:

POST /generic/verify_pin/

Request

```
{
    "pin": "1224",
    "uuid": "93e33277-87f7-417b-8ed2-371672b5297e"
}
```

Response

```
{
    "locked": false,
    "pin": "1224",
    "uuid": "93e33277-87f7-417b-8ed2-371672b5297e",
    "valid": false
}
```

Errors are handled much in the same way as `confirm_pin`. Calling this endpoint 5 times with the wrong PIN will lock the buyer. See *Locked State* for more information.

This change in state is the reason there is no *GET* endpoint for this API.

Reset

To start the reset flow, set the `needs_pin_reset` attribute on the buyer by patching the buyer:

PATCH /generic/buyer/int:id/

Request

```
{
    "needs_pin_reset": true
}
```

Response

Status Codes

- 202 Accepted – response processed.
- 404 Not Found – buyer not found.

Next you get the buyer's new pin and patch the buyer again:

PATCH /generic/buyer/int:id/

Request

```
{
    "new_pin": "8259"
}
```

Response

Status Codes

- 202 Accepted – response processed.
- 404 Not Found – buyer not found.

After these two steps you will use the `reset_confirm_pin` endpoint. It works the same way as the `confirm_pin` endpoint but instead checks against the buyer's `new_pin` rather than their `pin`:

Locked State

A buyer becomes locked when there have been 5 failed attempts to verify the PIN. Once the buyer is locked the verify PIN action will not be usable for 5 minutes. You can tell if a buyer is locked by checking the `pin_is_locked_out` property of the buyer data. Buyers that were locked out since the last time the PIN was changed or successfully verified will have the `pin_was_locked_out` property set to `true`.

Close

This does not delete the buyer, but does the following:

- calls a close signal the payment providers can listen to, in the case of Braintree it cancels all payment methods and subscriptions
- sets the account to inactive
- removes the email
- sets the uuid to something anonymous

Note: only if the close signal is succesfully processed will the account be set to inactive.

POST /generic/buyer/int:id/close/

Status Codes

- 204 No Content – account closed successfully
- 400 Bad Request – problem processing the uuid into a buyer
- 404 Not Found – active buyer not found, will trigger if you try to close an account twice
- 500 Internal Server Error – something went wrong with closing the account

The account is not deleted and can still be accessed by the URL to that account to preserve data integrity. For example:

- create a buyer with a POST to */generic/buyer*
- store the *resource_uri* in the response
- close the buyer with a POST to */generic/buyer/int:id/close*
- get the buyer with a GET to *resource_uri*, a *truncated* version of the response shows:

```
{
    "active": False,
    "email": "",
    "uuid": "anonymised-uuid:1b3db7a9-0e8f-43d8-b8da-b3317a147068"
}
```

1.5.2 Sellers

Sellers are identified by a UUID, which is a string (max 255 chars) that makes sense to the client. It must be unique within solitude, so we'd recommend prefixing the UUID, eg: *marketplace:<your-uuid>*

Sellers are added to solitude by a *POST* call. The POST should contain a unique UUID:

POST /generic/seller/

```
{
    "uuid": "acb21517-df02-4734-8173-176ece310bc1"
}
```

You can else get the details of a seller:

GET /generic/seller/9/

```
{
    "uuid": "acb21517-df02-4734-8173-176ece310bc1",
    "resource_uri": "/generic/seller/9/",
    "resource_pk": 16
}
```

1.5.3 Product

A product is a generic product that is being sold. To create a product specific payment provider, a generic product must first be created.

POST /generic/product/

Create a new product.

```
{
    "access": 1,
    "external_id": "external:5864962b-033e-4c7f-aabb-a3cd262e7042",
    "public_id": "product:279ae330-1c33-459d-b6ba-c22e5cba1c48",
    "secret": "some-secret",
    "seller": "/generic/seller/3/"
}
```

- `seller`: is a seller created with the *generic seller endpoint*.
- `external_id`: an id that corresponds to the sellers catalog.

- `public_id`: a publicly used id that will be used in the payment flow.
- `secret`: a generic back-end secret field, used for Paypal.
- `access`: either `1` seller will be used for purchasing or `2` seller can only be used for simulating payments.

GET /generic/product/id:int/
Get an existing product.


```
{
    "access": 1,
    "counter": "0",
    "created": "2015-02-05T12:41:50",
    "external_id": "external:5864962b-033e-4c7f-aabb-a3cd262e7042",
    "modified": "2015-02-05T12:41:50",
    "public_id": "product:279ae330-1c33-459d-b6ba-c22e5cba1c48",
    "resource_pk": 1,
    "resource_uri": "/generic/product/1/",
    "secret": "some-secret",
    "seller": "/generic/seller/3/",
    "seller_uuids": {
        "bango": null,
        "reference": null
    }
}
```


- `seller_uuids`: is a mapping of uuids for the specific payment providers.

1.5.4 Transaction

A transaction is created at the start of a payment through solitude. Its status is altered as the transaction is completed or cancelled as appropriate.

To iterate over the list of transactions:

GET /generic/transaction/

To get an individual transaction:

GET /generic/transaction/id:int/


```
{
    "amount": "0.62",
    "buyer": null,
    "created": "2013-04-15T05:39:22",
    "currency": "GBP",
    "notes": "",
    "pay_url": "https://provider.com/pay?transaction=1234",
    "provider": 1,
    "related": null,
    "relations": [],
    "resource_pk": 2977,
    "resource_uri": "/generic/transaction/2977/",
    "seller": "/generic/seller/385/",
    "seller_product": "/generic/product/449/",
    "status": 5,
    "type": 0,
    "uid_pay": "230450",
```

```
        "uid_support": "0",
        "uuid": "webpay:d8d143f3-d484-4903-bd29-bae3d280c5b3"
    }
```

Statuses:

- 0: `Pending` - when the transaction has started, the payment flow has been started and has been redirected on to the payment provider. For Bango, this is pretty much right away. This is the default.
- 1: `Completed` - the payment has been fully completed and processed.
- 2: `Checked` - the payment is in process and has been checked. This can be checked by a server to server notice (IPN for Paypal, Event Notification for Bango) or a manual transaction check. When checking to see if a transaction is successful, check to see if its `Completed` or `Checked`.
- 3: `Received` - we have received the transaction, but have not acted on it yet. This is an intermediate step between starting the transaction and passing it on to the payment provider. Bango does not use this.
- 4: `Failed` - an error occurred and the transaction failed.
- 5: `Cancelled` - the transaction was cancelled explicitly by the user.
- 6: `Started` - the calling application (e.g. webpay) has started preparing this transaction.
- 7: `Errored` - the calling application (e.g. webpay) was unable to complete creating the transaction because of an error.

To create a new transaction:

`POST /generic/transaction/`

```
    {
        "amount": "0.62",
        "buyer": null,
        "currency": "GBP",
        "notes": "",
        "pay_url": "https://provider.com/pay?transaction=1234",
        "provider": 1,
        "seller": "/generic/seller/385/",
        "seller_product": "/generic/product/449/",
        "source": "bango",
        "status": 5,
        "type": 0,
        "uid_pay": "230450",
        "uid_support": "0",
        "uuid": "webpay:d8d143f3-d484-4903-bd29-bae3d280c5b3"
    }
```

`GET /generic/transaction/id:int/`
Update an existing transaction.

```
    {
        "status_reason": "PROVIDER_LOOKUP_FAILURE"
    }
```

Note: not all fields can updated all the time, the ability to update a transaction is based upon logic within the transaction.

Only the following fields can be altered without limitation.

- `notes`

- `pay_url`
- `status_reason`
- `uid_pay`

Fields that can altered with limitation:

- `provider`: can be set, only if it is not set.
- `status`: see status notes below.

Status changes are limited in the following way:

- if a transaction was created before `settings.TRANSACTION_LOCKDOWN` then it cannot be altered.
- if a transaction is `Failed`, `Cancelled` or `Errored` its status cannot be altered.
- if a transaction is in `Checked` or `Received` it can only be moved to `Completed` or `Failed`.

1.6 Bango

For more information on the specific Bango APIs see: https://wiki.mozilla.org/Marketplace/BangoPayments

1.6.1 Sellers

The Bango tables contains the Bango specific data for that seller. First you'll need to create a Bango "package" by doing a POST call:

`POST /bango/package/`

Example successful seller creation:

```
{
    "seller": "/generic/seller/9",
    ...
}
```

1.6.2 Packages

A *GET* on a package will query the local solitude database about that package:

`GET /bango/package/9/`
Response

Example successful seller creation:

```
{
    "full": {},
    "created": "2013-01-30T09:41:34",
    "support_person_id": 232941,
}
```

The *full* field represents data polled from Bango. To get that information, send through *full* in the GET body. For example:

`GET /bango/package/9/`
Request

Example:

```
{
    "full": true
}
```

Response

Example successful seller creation:

```
{
    "full": {
        "vatNumber": null,
        "supportEmailAddress": "support@example.com",
        ...
    },
    "created": "2013-01-30T09:41:34",
    "support_person_id": 232941,
    ...
}
```

1.6.3 SBI Agreement

The SBI Agreement is 3 API calls (GetSBIAgreement, GetAcceptedSBIAgreement and AcceptSBIAgreement) rolled into one. You will need a valid Bango Seller in solitude to call this API:

GET /bango/sbi/agreement/
Response

Example successful SBI retrieval:

```
{
    "seller_bango": "/bango/package/29/"
}
```

This will return the text of the agreement and when the agreement will be valid for.

To set the agreement as approved:

POST /bango/sbi/
Response

Example successful SBI approved:

```
{
    "seller_bango": "/bango/package/29/"
}
```

This will return when the agreement was accepted and when it's valid too. The expiry date is also stored on the seller, so you can access that as well:

GET /bango/package/29/
Response

Example successful SBI retrieval:

```
{
    "sbi_expires": "2014-01-23"
}
```

If *sbi_expires* is empty, the agreement has not been approved.

1.6.4 Refunds

The refund API gives access to two Bango calls: "DoRefund" and "GetRefundStatus". You will need a valid payment transaction to start a refund.

POST /bango/refund/

Refund a payment.

Request

Parameters

- **uuid** – id of the payment transaction.

Example:

```
{
    "uuid": "uuid-of-the-payment-transaction"
}
```

Response

Status Codes

- 201 Created – refund processed. Examine the response contents to see the status of the refund and a pointer to the new refund.
- 400 Bad Request – there was a problem with the transaction chosen. Examine the response contents for more information.
- 404 Not Found – transaction not found at all.

Parameters

- **uuid** – the uuid of the transaction.
- **status** – the Bango response.
- **transaction** – the URL of the newly created transaction.

Example successful refund:

```
{
    "fake_response": null,
    "resource_pk": 2,
    "resource_uri": "/bango/refund/2/",
    "status": "OK",
    "transaction": "/generic/transaction/2/",
    "uuid": "sample:uid"
}
```

GET /bango/refund/status/

Look up the status of refund.

Note: If the response from Bango is different from the transaction state, then the transaction is updated to reflect the refund's new status. This might happen for PENDING refunds.

Request

Parameters

- **uuid** – uuid of the refund transaction.

Example:

```
{
    "uuid": "sample:uid"
}
```

Response

Status Codes

- 200 OK – successfully completed.

Parameters

- **status** – the Bango response.
- **transaction** – the URL of the refund transaction.

```
{
    "fake_response": null,
    "resource_pk": 1,
    "resource_uri": "/bango/refund/1/",
    "status": "OK",
    "transaction": "/generic/transaction/1/"
}
```

1.7 Braintree

Make sure your *Braintree settings* are up to date so that Solitude can connect to the API.

1.7.1 Tokens

Calls braintree ClientToken.generate:

POST /braintree/token/generate/

Response JSON Object

- **token** (*string*) – the token returned by Braintree.

Status Codes

- 200 OK – token successfully generated.

1.7.2 Customers

Creates a customer in Braintree and the corresponding buyer and Braintree buyer in solitude. If the buyer exists in solitude already, it is not created. If the customer exists in Braintree already, it is not created.

POST /braintree/customer/

Request JSON Object

- **uuid** (*string*) – the UUID of the *buyer*.

```
{
    "braintree": {
        "created_at": "2015-05-06T15:35:41.519",
        "id": "customer-id",
        "updated_at": "2015-05-06T15:35:41.519"
```

```
        },
        "mozilla": {
            "active": true,
            "braintree_id": "customer-id",
            "buyer": "/generic/buyer/3/",
            "counter": 0,
            "created": "2015-05-06T15:35:41.523",
            "id": 2,
            "modified": "2015-05-06T15:35:41.523",
            "resource_pk": 2,
            "resource_uri": "/braintree/mozilla/buyer/2/"
        }
    }
```


Response JSON Object

- **braintree id** (*string*) – id.
- **braintree created_at** (*string*) – created date and time.
- **braintree updated_at** (*string*) – updated date and time.

Status Codes

- 201 Created – customer and Braintree buyer successfully created.

Data stored in solitude

Some information is stored in solitude after creating a customer.

GET /braintree/mozilla/buyer/<buyer id>/

Request JSON Object

- **uuid** (*string*) – the uuid of the buyer in solitude.
- **nonce** (*string*) – the payment nonce returned by Braintree.


```
    {
        "active": true,
        "braintree_id": "customer-id",
        "buyer": "/generic/buyer/3/",
        "counter": 0,
        "created": "2015-05-06T15:35:41.523",
        "id": 2,
        "modified": "2015-05-06T15:35:41.523",
        "resource_pk": 2,
        "resource_uri": "/braintree/mozilla/buyer/2/"
    }
```


Response JSON Object

- **active** (*boolean*) – if the buyer is currently active.
- **braintree_id** (*string*) – the id of the customer on Braintree. This field is read only.
- **buyer** (*string*) – URI to *a buyer object*

PATCH /braintree/mozilla/buyer/<buyer id>/

Request JSON Object

- **active** (*boolean*) – if the buyer is currently active.

GET /braintree/mozilla/buyer/

Query Parameters

- **buyer** – the primary key of the buyer.
- **active** – the active status.

1.7.3 Payment Methods

Create or update a payment method in Braintree and the corresponding payment method in solitude.

POST /braintree/paymethod/

Request JSON Object

- **buyer_uuid** (*string*) – the uuid of the buyer in solitude.
- **nonce** (*string*) – the payment nonce returned by Braintree.


```
{
    "braintree": {
        "created_at": "2015-05-05T14:22:26.650",
        "token": "da-token",
        "updated_at": "2015-05-05T14:22:26.650"
    },
    "mozilla": {
        "active": true,
        "braintree_buyer": "/braintree/mozilla/buyer/16/",
        "counter": 0,
        "created": "2015-05-05T14:22:26.656",
        "id": 4,
        "modified": "2015-05-05T14:22:26.656",
        "provider_id": "da-token",
        "resource_pk": 4,
        "resource_uri": "/braintree/mozilla/paymethod/4/",
        "truncated_id": "7890",
        "type": 1,
        "type_name": "visa"
    }
}
```


Response JSON Object

- **braintree token** (*string*) – id of the payment method in Braintree.
- **braintree created_at** (*string*) – created date and time.
- **braintree updated_at** (*string*) – updated date and time.

Status Codes

- 201 Created – payment method created.

Delete a payment method. This will delete the payment method in Braintree and is not reversible.

POST /braintree/paymethod/delete/

Request JSON Object

- **paymethod** (*string*) – the resource_uri of the payment method in solitude.

Status Codes

- 204 No Content – payment method deleted.

Data stored in solitude

Some information about the payment method is stored in solitude.

GET /braintree/mozilla/paymethod/<method id>/

```
{
  "active": true,
  "braintree_buyer": "/braintree/mozilla/buyer/2/",
  "counter": 0,
  "created": "2015-05-05T14:25:38",
  "id": 1,
  "modified": "2015-05-05T14:25:38",
  "provider_id": "da-token",
  "resource_pk": 1,
  "resource_uri": "/braintree/mozilla/paymethod/1/",
  "truncated_id": "some",
  "type": 1,
  "type_name": "visa"
}
```

Response JSON Object

- **active** (*boolean*) – active flag for the method.
- **braintree_buyer** (*string*) – URI to *a braintree buyer object*.
- **provider_id** (*string*) – an id for the payment method on the provider, this field is read only.
- **truncated_id** (*string*) – a truncated id of the payment type, for example for a credit card, the last 4 digits, this field is read only.
- **type** (*int*) – *1* for credit card is currently the only one supported, this field is read only.
- **type_name** (*string*) – name of the type of purchase, this field is read only.

GET /braintree/mozilla/paymethod/

Query Parameters

- **braintree_buyer** – the primary key of the braintree_buyer.
- **braintree_buyer__buyer__uuid** – the uuid for the buyer.
- **active** – the active status.

1.7.4 Subscriptions

Create a subscription (including recurring donation plans) in Braintree and the corresponding subscription in solitude.

POST /braintree/subscription/

Request JSON Object

- **paymethod** (*string*) – the uri of the payment method.

- **plan** (*string*) – the braintree ID of the plan being purchased. This must also match *public_id* of *Product* and the ID must link to a known, configured product.
- **amount** (*string*) – custom payment amount as a decimal string. This only applies to subscription plans that allow custom amounts such as donations.


```
{
    "braintree": {
        "created_at": "2015-05-06T13:34:53.746",
        "id": "some:id",
        "updated_at": "2015-05-06T13:34:53.746"
    },
    "mozilla": {
        "active": true,
        "counter": 0,
        "created": "2015-05-06T13:34:53.763",
        "id": 1,
        "amount": null,
        "modified": "2015-05-06T13:34:53.763",
        "paymethod": "/braintree/mozilla/paymethod/1/",
        "provider_id": "some:id",
        "resource_pk": 1,
        "resource_uri": "/braintree/mozilla/subscription/1/",
        "seller_product": "/generic/product/1/"
    }
}
```


Response JSON Object

- **braintree id** (*string*) – id of the subscription in braintree.
- **braintree created_at** (*string*) – created date and time.
- **braintree updated_at** (*string*) – updated date and time.

Status Codes

- 201 Created – payment method created.

Change payment method on a subscription:

POST /braintree/subscription/paymethod/change/

Request JSON Object

- **paymethod** (*string*) – the resource_uri of the paymethod in solitude.
- **subscription** (*string*) – the resource_uri of the subscription in solitude.

Status Codes

- 200 OK – subscription changed.

Cancel a subscription. This will cancel the subscription in Braintree and is not reversible.

POST /braintree/subscription/cancel/

Request JSON Object

- **subscription** (*string*) – the resource_uri of the subscription in solitude.

The response is in the same format as for creation.

Status Codes

- 200 OK – subscription cancelled.

Data stored in solitude

Some information about the subscripton is stored in solitude.

GET /braintree/mozilla/subscription/<subscription id>/

```
{
    "active": true,
    "counter": 0,
    "created": "2015-05-01T18:21:49",
    "id": 1,
    "paymethod": "/braintree/mozilla/paymethod/2/",
    "modified": "2015-05-01T18:21:49",
    "provider_id": "some:id",
    "resource_pk": 1,
    "resource_uri": "/braintree/mozilla/subscription/1/",
    "seller_product": "/generic/product/2/"
}
```

Response JSON Object

- **active** (*boolean*) – active flag for the method.
- **provider_id** (*string*) – an id for the subscription on the provider, this field is read only.
- **paymethod** (*string*) – the URI to a payment object.
- **seller_product** (*string*) – the URI to a seller product.

PATCH /braintree/mozilla/subscription/<subscription id>/

Request JSON Object

- **active** (*boolean*) – if the subscription is currently active.

GET /braintree/mozilla/subscription/

Query Parameters

- **active** – if the subscription is active.
- **paymethod** – the primary key of the payment method.
- **paymethod__braintree_buyer** – the primary key of the braintree buyer.
- **paymethod__braintree_buyer__buyer** – the primary key of the buyer.
- **provider_id** – the plan id for this subscription.
- **seller_product** – the primary key of the product.

1.7.5 Sale

A sale is a one off payment to call the Braintree Transaction API. For more information see the Braintree transaction documentation.

This should not be used for subscriptions.

POST /braintree/sale/

Request JSON Object

- **amount** – the amount of the transaction, within the maximum and minimum limits
- **product_id** – the product_id as defined by payments-config.
- **nonce** – (optional) the payment nonce returned by Braintree, used when no payment method is stored.
- **paymethod** – (optional) the URI to a payment object.


```
{
  "mozilla": {
    "generic": {
      "resource_pk": 1,
      "related": null,
      "seller_product": "/generic/product/1/",
      "currency": "USD",
      "uid_pay": null,
      "uuid": "",
      "uid_support": "test-id",
      "relations": [],
      "seller": "/generic/seller/1/",
      "source": null,
      "provider": 4,
      "pay_url": null,
      "type": 0,
      "status": 2,
      "buyer": null,
      "status_reason": null,
      "created": "2015-08-17T19:17:04.296",
      "notes": null,
      "amount": "5.00",
      "carrier": null,
      "region": null,
      "resource_uri": "/generic/transaction/1/"
    },
    "braintree": {
      "kind": "",
      "transaction": "/generic/transaction/1/",
      "next_billing_period_amount": null,
      "created": "2015-08-17T19:17:04.298",
      "paymethod": null,
      "counter": 0,
      "billing_period_end_date": null,
      "modified": "2015-08-17T19:17:04.298",
      "next_billing_date": null,
      "resource_pk": 1,
      "resource_uri": "/braintree/mozilla/transaction/1/",
      "billing_period_start_date": null,
      "id": 1,
      "subscription": null
    }
  },
  "braintree": {}
}
```


Response JSON Object

- **mozilla.generic** – the generic transaction object.

- **mozilla.generic.buyer** – this will be the buyer or empty if no buyer is registered.
- **mozilla.braintree** – the braintree transaction object.

Notes: * either a *nonce* or a *paymethod* must exist, but not both

1.7.6 Webhook

When Braintree completes certain actions, they will make a request to the configured webhook URL. That will be payments-service which then passes it on to this endpoint. For more information see the Braintree webhook documentation.

GET /braintree/webhook/

Query Parameters

- **string** (*bt_challenge*) – the bt_challenge issued by Braintree.

Response JSON Object

- **string** – a token returned by the Braintree verify API.

Status Codes

- 200 OK – token verified and returned.

POST /braintree/webhook/

Request JSON Object

- **bt_signature** – the bt_signature issued by Braintree.
- **bt_payload** – the bt_payload issued by Braintree.


```
    {
      "mozilla": {
        "buyer": {
          "active": true,
          "email": "email@example.com",
          "needs_pin_reset": false,
          "new_pin": false,
          "pin": false,
          "pin_confirmed": false,
          "pin_failures": 0,
          "pin_is_locked_out": false,
          "pin_was_locked_out": false,
          "resource_pk": 32,
          "resource_uri": "/generic/buyer/32/",
          "uuid": "dc728c67-bcf8-4237-962d-cb15b2916e21"
        },
        "paymethod": {
          "resource_pk": 22,
          "resource_uri": "/braintree/mozilla/paymethod/22/",
          "braintree_buyer": "/braintree/mozilla/buyer/31/",
          "id": 22,
          "created": "2015-06-16T18:03:43.902",
          "modified": "2015-06-16T18:03:43.902",
          "counter": 0,
          "active": true,
          "provider_id": "29e66c1b-6824-4a41-80d2-fa58ec8fb206",
          "type": 1,
```

```
      "type_name": "",
      "truncated_id": ""
    },
    "subscription": {
      "resource_pk": 12,
      "resource_uri": "/braintree/mozilla/subscription/12/",
      "paymethod": "/braintree/mozilla/paymethod/22/",
      "seller_product": "/generic/product/18/",
      "id": 12,
      "created": "2015-06-16T18:03:43.904",
      "modified": "2015-06-16T18:03:43.904",
      "counter": 0,
      "active": true,
      "provider_id": "some-bt:id"
    },
    "transaction": {
      "generic": {
        "amount": "10",
        "buyer": "/generic/buyer/32/",
        "carrier": null,
        "created": "2015-06-16T18:03:43.915",
        "currency": "USD",
        "notes": null,
        "pay_url": null,
        "provider": 4,
        "region": null,
        "related": null,
        "relations": [],
        "resource_pk": 7,
        "resource_uri": "/generic/transaction/7/",
        "seller": "/generic/seller/19/",
        "seller_product": "/generic/product/18/",
        "source": null,
        "status": 2,
        "status_reason": "settled",
        "type": 0,
        "uid_pay": null,
        "uid_support": "bt:id",
        "uuid": "f424e706-9c17-4d6a-9287-e6db28e46ec6"
      },
      "braintree": {
        "resource_pk": 5,
        "resource_uri": "/braintree/mozilla/transaction/5/",
        "paymethod": "/braintree/mozilla/paymethod/22/",
        "subscription": "/braintree/mozilla/subscription/12/",
        "transaction": "/generic/transaction/7/",
        "id": 5,
        "created": "2015-06-16T18:03:43.916",
        "modified": "2015-06-16T18:03:43.916",
        "counter": 0,
        "billing_period_end_date": "2015-07-15T18:03:43.904",
        "billing_period_start_date": "2015-06-16T18:03:43.904",
        "kind": "subscription_charged_successfully",
        "next_billing_date": "2015-07-16T18:03:43.904",
        "next_billing_period_amount": "10"
      }
    },
    "product": {
```

```
        "seller": "/generic/seller/19/",
        "access": 1,
        "resource_uri": "/generic/product/18/",
        "resource_pk": 18,
        "secret": null,
        "seller_uuids": {
          "bango": null,
          "reference": null
        },
        "public_id": "brick",
        "external_id": "3089c93d-eb16-4233-83d3-37653369ff8c"
      }
    },
    "braintree": {
      "kind": "subscription_charged_successfully"
    }
  }
```


Note: some webhooks (such as *subscription_canceled*) may not contain a transaction. If that's the case then the *mozilla.transaction* and *mozilla.paymethod* fields will be empty.

Response JSON Object

- **mozilla.buyer** – a *buyer*.
- **mozilla.paymethod** – a *payment method* (optional).
- **mozilla.product** – a *product*
- **mozilla.subscription** – a *subscription*.
- **mozilla.transaction.generic** – a *generic transaction* (optional).
- **mozilla.transaction.braintree** – a *braintree transaction* (optional).
- **braintree.kind** – the kind of webhook.
- **braintree.next_billing_period_amount** – the amount of the next charge.
- **braintree.next_billing_date** – the date of the next charge.

Status Codes

- 200 OK – webhook parsed successfully, solitude may have acted on the webhook and is returning data with the expectation that the client will as well.
- 204 No Content – webhook parsed successfully, however solitude did not act on the webhook and does not expect the caller to act either.

1.7.7 Transaction

The webhook returns transaction details to solitude. Solitude then creates a generic transaction object. It also creates a Braintree transaction that contains some information about the purchase transaction.

GET /braintree/mozilla/transaction/<transaction id>/
Get a single braintree transaction object.


```
    {
        "id": 1,
        "billing_period_end_date": "2015-07-10T12:20:19.926",
        "billing_period_start_date": "2015-06-11T12:20:19.926",
        "created": "2015-06-11T12:20:19.926",
```

```
        "counter": 0,
        "kind": "disbursement",
        "modified": "2015-06-11T12:20:19.926",
        "next_billing_date": "2015-07-11T12:20:19.926",
        "next_billing_period_amount": "10.00",
        "paymethod": "/braintree/mozilla/paymethod/2/",
        "resource_pk": 1,
        "resource_uri": "/generic/transaction/1/",
        "subscription": "/braintree/mozilla/subscription/2/",
        "transaction": "/generic/transaction/2/"
    }
```

Response JSON Object

- **paymethod** (*string*) – the URI to a payment object.
- **subscription** (*string*) – the URI to a subscription object.
- **transaction** (*string*) – the URI to a transaction object.

The fields *kind*, *next_billing_date*, *next_billing_period_amount*, *billing_period_end_date*, *billing_period_start_date* are copies of the data from Braintree. Please see the Braintree documentation for more information.

GET /braintree/mozilla/transaction/
Get all braintree transactions.

Query Parameters

- **transaction__buyer__uuid** – only get transactions belonging to this *generic buyer* UUID.

1.7.8 Development Tips

When developing on systems that rely on Braintree data in Solitude you can reset some data with the `./manage.py braintree_reset` script. See the `--help` output for details.

1.8 Zippy

Documentation on zippy and reference implementation.

Note that in the following examples `{*uuid}` refers to an actual `uuid`.

1.8.1 Sellers

POST /provider/reference/sellers/
Create a seller.

Request

Parameters

- **uuid** – uuid of the seller.
- **email** – email of the seller.
- **name** – name of the seller.

- **status** – status of the seller.
- **seller** – the url to the seller as returned from seller creation.

Example:

```
{
    "seller": "{seller-url}",
    "email": "jdoe@example.org",
    "name": "John",
    "status": "ACTIVE",
    "uuid": "{seller-uuid}"
}
```

Response

Status Codes

- 201 Created – seller created. Examine the response contents to see the status of the seller and a pointer to the new seller.
- 400 Bad Request – there was a problem with the seller creation. Examine the response contents for more information.

See retrieving a seller for detail on the response.

GET /provider/reference/sellers/{seller-uuid}/
Retrieve a seller.

Response

Status Codes

- 200 OK – seller retrieved. Examine the response contents to see the status of the seller and a pointer to the seller.
- 400 Bad Request – there was a problem with the seller retrieval. Examine the response contents for more information.

Parameters

- **email** – email of the seller.
- **name** – name of the seller.
- **status** – status of the seller.
- **agreement** – an optional date that can be used for terms validation.
- **resource_name** – the name of the resource.
- **id** – the primary key of the resource.
- **resource_uri** – the URI of the resource.
- **uuid** – a UUID for the resource.
- **seller** – the URI of the generic seller.

Example successful seller retrieval:

```
{
    "agreement": "",
    "email": "jdoe@example.org",
    "name": "John",
    "resource_name": "sellers",
    "id": "{seller-id}",
```

```
        "resource_uri": "/provider/reference/sellers/{seller-id}",
        "status": "ACTIVE",
        "seller" "/generic/seller/1/",
        "uuid": "{seller-uuid}"
    }
```

PUT /provider/reference/sellers/{seller-uuid}/
Update a seller.

Request

All parameters are optionals.

Parameters

- **uuid** – uuid of the seller.
- **email** – email of the seller.
- **name** – name of the seller.
- **status** – status of the seller.

Example:

```
    {
        "name": "Jack"
    }
```

Response

Status Codes

- 201 Created – seller created. Examine the response contents to see the status of the seller and a pointer to the seller.
- 400 Bad Request – there was a problem with the seller modification. Examine the response contents for more information.

Parameters

- **email** – email of the seller.
- **reference** – the contents of the response from the reference server.
- **> name** (*reference*) – name of the seller.
- **> status** (*reference*) – status of the seller.
- **> agreement** (*reference*) – an optional date that can be used for terms validation.
- **> resource_name** (*reference*) – the name of the resource.
- **> id** (*reference*) – the primary key of the resource.
- **resource_uri** – the URI of the resource.

Example successful seller modification:

```
    {
        "id": "{seller-uuid}",
        "reference": {
            "agreement": "",
            "email": "jdoe@example.org",
            "name": "Jack",
            "resource_name": "sellers",
```

```
            "resource_uri": "{seller-uri}",
            "id": "{seller-uuid}",
            "status": "ACTIVE"
        },
        "resource_uri": "/sellers/{seller-uuid}",
        "seller": "{seller-uri}"
    }
```

1.8.2 Products

Using that newly created "seller", we can now create a "product".

POST /provider/reference/products/

Create a product.

Request

Parameters

- **name** – name of the product.
- **seller_product** – url of the generic product.
- **seller_reference** – url of the reference seller.
- **uuid** – a uuid for this product.

Example:

```
    {
        "name": "Product name",
        "uuid": "{product-uuid}",
        "seller_product": "{seller-product-url}",
        "seller_reference": "{seller-reference-url}"
    }
```

Response

Status Codes

- 201 Created – product created. Examine the response contents to see the status of the product and a pointer to the new product.
- 400 Bad Request – there was a problem with the product creation. Examine the response contents for more information.

Parameters

- **id** – the primary key of the resource.
- **seller_product** – URI of the generic product.
- **seller_reference** – URI of the reference seller.
- **reference** – the contents of the response from the reference server.
- **> external_id** (*reference*) – the external id.
- **> name** (*reference*) – name of the product.
- **> resource_name** (*reference*) – the name of the resource.
- **> resource_uri** (*reference*) – the URI of the resource.

- **> seller_id** (*reference*) – uuid of the seller.
- **> status** (*reference*) – status of the product.
- **> uuid** (*reference*) – the uuid of the product.

Example successful product creation:

```
{
    "id": "{product-id}",
    "reference": {
        "external_id": "{external-uuid}"
        "name": "Product name",
        "resource_name": "products",
        "resource_uri": "/products/reference/{product-id}",
        "seller_id": "{seller-uuid}",
        "status": "ACTIVE",
        "uuid": "{product-uuid}"
    },
    "resource_uri": "/products/reference/{product-id}",
    "seller_product": "{seller-product-url}",
    "seller_reference": "{seller-reference-url}",
}
```

1.8.3 Transactions

Let's buy that product by creating a "transaction".

POST /provider/reference/transactions/

Create a transaction.

Request

Parameters

- **carrier** – the carrier of the transaction.
- **currency** – the currency of the transaction.
- **price** – the price of the transaction.
- **product_id** – uuid of the product.
- **ext_transaction_id** – uuid of the transaction.
- **pay_method** – the payment method of the transaction.
- **region** – the region concerned by the transaction.
- **error_url** – the URL to reach in case of error of the transaction.
- **success_url** – the URL to reach in case of success of the transaction.

Example:

```
{
    "carrier": "USA_TMOBILE",
    "currency": "EUR",
    "price": "0.99",
    "product_id": "{product-uuid}",
    "error_url": "http://marketplace.firefox.com/mozpay/provider/error/",
    "success_url": "http://marketplace.firefox.com/mozpay/provider/success/",
    "ext_transaction_id": "{transaction-uuid}",
```

```
    "pay_method": "OPERATOR",
    "region": "123"
}
```

Response

Status Codes

- 201 Created – transaction created. Examine the response contents to see the status of the transaction and the token.
- 400 Bad Request – there was a problem with the transaction creation. Examine the response contents for more information.

Parameters

- **carrier** – the carrier of the transaction.
- **currency** – the currency of the transaction.
- **price** – the price of the transaction.
- **product_id** – uuid of the product.
- **ext_transaction_id** – uuid of the transaction.
- **pay_method** – the payment method of the transaction.
- **region** – the region concerned by the transaction.
- **error_url** – the URL to reach in case of error of the transaction.
- **success_url** – the URL to reach in case of success of the transaction.
- **resource_name** – the name of the resource.
- **id** – the primary key of the resource.
- **resource_uri** – the URI of the resource.
- **status** – status of the transaction. Should be STARTED at this point.
- **token** – the security token for the transaction.

Example successful transaction creation:

```
{
    "carrier": "USA_TMOBILE",
    "currency": "EUR",
    "product_id": "{product-uuid}",
    "error_url": "http://marketplace.firefox.com/mozpay/provider/error/",
    "success_url": "http://marketplace.firefox.com/mozpay/provider/success/",
    "ext_transaction_id": "{transaction-uuid}",
    "pay_method": "OPERATOR",
    "price": "0.99",
    "region": "123"
    "resource_name": "transactions",
    "id": "{product-uuid}",
    "resource_uri": "/transactions/{product-uuid}",
    "token": "97ccb8ced0318a2751e936e354848...",
    "status": "STARTED"
}
```

1.8.4 Terms Agreement

GET /provider/reference/sellers/{seller-uuid}/
Retrieve terms related to a given seller.

Response

Status Codes

- 200 OK – terms retrieved. Examine the response contents to see the content of the terms and an agreement date.
- 400 Bad Request – there was a problem with the terms retrieval. Examine the response contents for more information.

Parameters

- **terms** – the text containing terms, can be lengthy.
- **agreement** – the datetime of the agreement of the terms by the user.

Example successful terms retrieval:

```
{
    "terms": "Terms for seller: John...",
    "agreement": "2013-11-19T11:48:49.158Z"
}
```

1.9 Proxy

To add a further layer of security, Solitude can be run in two modes:

- *standalone*: in this case Solitude is only run with **one instance** and communicates to the payment providers (PayPal, Bango etc). That one instance knows everything about how to interact with the database and has the appropriate settings for communicating with those providers.
  - Requests go: **client > solitude > provider**.
- *proxy*: in this case Solitude is run with **two instances**, a database server and proxy server.
  - Requests go: **client > solitude database server > solitude proxy server > provider**.
  - *database server* this **can** read and write to the database, the cache and so on. It **cannot** talk to the provider. It has no provider credentials.
  - *proxy server* this **cannot** read and write to the database, the cache and so on. It **can** talk to provider. It has the provider credentials.

By default solitude runs in standalone mode. Running using *runserver* or the *wsgi/playdoh.py* script will run in this mode.

Running both the database and proxy server on the same instance might not give you much of an advantage. The intention is to run them in seperate servers and have appropriate security between them.

To run in proxy mode, make the following changes:

- *database server* ensure you have not specified any sensitive provider settings.
  - For Bango set *BANGO_PROXY* to point to the *proxy server* referencing the path */proxy/bango* for example:

```
BANGO_PROXY = 'https://some.server.local/proxy/bango'
```

  - For payment providers using *zippy*, set *ZIPPY_PROXY* to point to the *proxy server* referencing the path */proxy/provider* for example:

```
BANGO_PROXY = 'https://some.server.local/proxy/provider'
```

    You should also ensure that you do not have the *auth* section in your *ZIPPY_CONFIGURATION* dictionary, since the *database server* will not be referencing the *auth*.

- *proxy server* ensure you have not specified any database or cache settings, but have specified the provider settings, such as username, password, sandbox and so on.
  - For zippy, ensure that the *ZIPPY_CONFIGURATION* configuration has the *auth* dictionary and *url* string.

To run the proxy server, run with the environment variable:

```
SOLITUDE_PROXY='enabled'
```

To run as a wsgi file, just use *wsgi/proxy.py* and it will set this variable for you.

1.9.1 Errors

If the proxy encounters a response code that is not a 2xx code then it will log a warning that there might be issues for the db instance. Example:

```
s.proxy:ERROR Warning response status: 404
```

In this case it's up to the client to deal with the issue before continuing processing. It might be acceptable for the API to return a 404 or 302 and the client knows how to deal with that.

If the client has a fatal error:

```
s.proxy:ERROR ConnectionError: [Errno 8] nodename nor servname provided, or not known
```

This will return a response of 500 to the calling library. It's that libraries job to cope with the 500 errors. In this case the Bango client in solitude detects 500 and raises a *ProxyError*:

```
File "/Users/andy/sandboxes/solitude/lib/bango/client.py\", line 133, in send
    raise ProxyError(msg)
        "type": "<class 'lib.bango.errors.ProxyError'>",
        "value": "Proxy returned: 500 from: https://webservices.test.bango.org/mozillaexporter/servic
```

1.9.2 Testing

When solitude with proxy is setup, run a test command against the service.

For Bango, from the command line, with all the solitude requirements installed:

```
cd samples
python bango-basic.py
```

If you've got an error talking to Bango you'll get a proxy error as outlined above.

1.10 Services

These are resources to provide information to clients about the status.

GET /services/request/
Echoes information back about the requet.

Response

Parameters

- **authenticated** – the OAuth key used to authenticate.

Status Codes

- 200 OK – successful.

GET /services/status/
Returns information about things solitude needs. Useful for nagios.

Response

Example:

```
{
    "meta":
    {
        "limit": 20,
        "next": null,
        "offset": 0,
        "previous": null,
        "total_count": 1
    },
    "objects":
    [{
        "cache": true,
        "db": true,
        "resource_uri": "",
        "settings": true
    }]
}
```

Status Codes

- 200 OK – successful.
- 500 Internal Server Error – theres a problem on the server.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

/bango

/braintree

/generic

/provider

/services