

---

# **webpay Documentation**

***Release 1.0***

**Mozilla**

October 20, 2015



<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Developers . . . . .	3
1.2	Webpay API . . . . .	5
1.3	Solitude API Client . . . . .	8
1.4	Localization Testing . . . . .	11
1.5	Services . . . . .	12
1.6	Flows . . . . .	13
<b>2</b>	<b>Indices and tables</b>	<b>15</b>
	<b>HTTP Routing Table</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>



Webpay is an implementation of the [WebPaymentProvider](#) spec. It hosts the payment flow inside `navigator.mozPay()` when making app purchases or in-app payments on Firefox OS.

This guide can help you do a few things:

- *Install and configure* your own WebPay server for development.
- Understand the APIs WebPay consumes and generally how things work.

This is also available as a [PDF](#).

The section on *using a hosted webpay* has moved to the [payments section](#) of the Marketplace documentaion .



## 1.1 Developers

Hello, developers! This will get you set up with a local WebPay server. You can also *use a hosted* WebPay server.

### 1.1.1 Install With Docker

The easiest way to set up Webpay and all of its dependencies is to install everything with Docker according to the [marketplace-env](#) instructions.

### 1.1.2 Install Manually

You need Python 2.7, and MySQL, and a few NodeJS commands like [stylus](#) for minifying JS/CSS. Install system requirements with **‘homebrew’** (Mac OS X):

```
brew tap homebrew/versions
brew install python mysql swig
```

To develop locally you also need:

- An instance of the [Solitude](#) payment API running. If you run it with mock services (such as `BANGO MOCK=True`) then some things will still work. You can configure `webpay` with `SOLITUDE_URL` pointing at your localhost.
- Access to the [Zamboni](#) db. For extra points this can be a read only slave. You can configure `zamboni` with `MARKETPLACE_URL` pointing at your localhost.

Let’s install `webpay`! Clone the source:

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

Install all Python dependencies. You probably want to do this within a [virtualenv](#). If you use [virtualenvwrapper](#) (recommended) set yourself up with:

```
mkvirtualenv --python=python2.7 webpay
```

Install with:

```
pip install --no-deps -r requirements/dev.txt --find-links https://pyrepo.addons.mozilla.org/
```

Out of the box, webpay makes some assumptions in the settings file and should not need a custom settings files. Some environment variables are configurable from the environment, they are: `ZAMBONI_URL`, `SOLITUDE_URL`. See the [marketplace docs](#) for information on the environment variables and how they affect the services.

You can now fire up a development server:

```
python manage.py runserver 0.0.0.0:8001
```

Try it out at <http://localhost:8001/mozpay/> . If you see a form error about a missing JWT then you are successfully up and running.

If you can't log in with Persona check the value of `SITE_URL` in your local settings. It must match the URL bar of how you run your dev server exactly.

See *this section* for how to set up a B2G device to talk to your brand new local development server.

### 1.1.3 Setting Up the Tests

You will need to install the python testing dependencies for python or UI testing:

```
pip install -r requirements/test.txt
```

### 1.1.4 Running Tests

Webpay has integration tests that make HTTP requests to Django views or test public functions and classes directly. You can run the test suite like this:

```
python manage.py test
```

### 1.1.5 Building the Docs

To build these very docs that you are reading while developing locally, do this from your webpay root:

```
pip install -r requirements/docs.txt
make -C docs/ html
```

Then open `docs/_build/html/index.html` in a browser.

### 1.1.6 Overriding JS settings from Django settings

JS settings are overridden from the `webpay.settings.base.JS_SETTINGS` dict.

Here's an example to override a setting `foo` with the value `True`:

```
base.JS_SETTINGS['foo'] = True
```

### 1.1.7 Using JWTs for development

Each payment begins with a JWT (Json Web Token) so you'll need to start with a JWT if you want to see the complete payment flow. The best way to get a valid JWT is to make a real purchase using your local Marketplace or any app that has a valid in-app payment key. When you start a purchase from B2G check your B2G console. In stdout you should see a link that you can copy and paste into a browser to use better dev tools. Here is an example of what that looks like:

`http://localhost:8001/mozpay/?req=eyJhbGciOiAiSFMyNTYiLCJhdHlwIjogIkpXVCJ9.eyJhdWQiOiAibG9jYXob3N0I`

## 1.1.8 Displaying statsd results

You can configure your `webpay/settings/local.py` settings to visualize the summary table generated by `django-statsd` counting the number of keys logged and the time spent in views:

```
NOSE_PLUGINS = [
    'nosenicedots.NiceDots',
    'django_statsd.NoseStatsd',
]
NOSE_ARGS = [
    '--logging-clear-handlers',
    '--with-statsd',
]
STATSD_CLIENT = 'django_statsd.clients.nose'
```

## 1.2 Webpay API

Webpay provides a REST API for clients to interact with the server.

All API's use JSON for request and responses.

### 1.2.1 PIN

The PIN API lets you check, create and update the PIN through Webpay.

---

**Note:** This API Requires authentication through Persona prior to access.

---

#### **GET** `/mozpay/v1/api/pin/`

Returns information about the PIN for the current user. Will not return the actual PIN.

#### **Response**

Example:

```
{
  "pin": true,
  "pin_locked_out": null,
  "pin_is_locked_out": false,
  "pin_was_locked_out": false
}
```

#### **Status Codes**

- **200** – successfully completed.
- **403** – not authenticated.

#### **Parameters**

- **pin** (*boolean*) – if a PIN exists or not
- **pin\_locked\_out** (*date time*) – if a PIN is locked out, this is when it occurred
- **pin\_is\_locked\_out** (*boolean*) – if a PIN is locked out

- **pin\_was\_locked\_out** (*boolean*) – if a PIN has been locked out

**POST /mozpay/v1/api/pin/**

Creates a PIN for the current user.

**Request**

**Parameters**

- **pin** (*string*) – 4 numbers in the range 0-9 as a string

**Response**

**Status Codes**

- **204** – successfully created.
- **400** – invalid form data.
- **403** – not authenticated.

**PATCH /mozpay/v1/api/pin/**

Updates a PIN for the current user.

**Request**

**Parameters**

- **pin** (*string*) – 4 numbers in the range 0-9 as a string

**Response**

**Status Codes**

- **204** – successfully updated.
- **400** – invalid form data.
- **403** – not authenticated.

## 1.2.2 Pin Check

**POST /mozpay/v1/api/pin/check/**

Checks a posted PIN against a stored pin.

**Request**

**Parameters**

- **pin** (*string*) – 4 numbers in the range 0-9 as a string

**Response**

Example:

```
{
  "pin": true,
  "pin_locked_out": null,
  "pin_is_locked_out": null,
  "pin_was_locked_out": null
}
```

**Status Codes**

- **200** – successfully completed.

- **400** – incorrect PIN.
- **403** – not authenticated.
- **404** – no user exists.

The response is the same as for the PIN API.

### 1.2.3 Pay

The Pay API lets you start a purchase.

**POST** /mozpay/v1/api/pay/

Start a purchase.

#### Request

##### Parameters

- **req** (*str*) – the JWT request for starting a payment
- **mnc** (*str*) – the MNC (mobile network code) for the device (optional)
- **mcc** (*str*) – The MCC (mobile country code) for the device (optional)

#### Response

##### Parameters

- **status** (*str*) – “ok” if successful
- **simulation** (*dict*) – Indicates the type of simulated payment. If this is a normal payment, not a simulation, it will be `False`. Otherwise it will be one of the [valid simulation results](#) such as `{"result": "postback"}`.

```
{
  "status": "ok",
  "simulation": {"result": "postback"}
}
```

##### Status Codes

- **200** – successful.
- **400** – invalid form data.

**GET** /mozpay/v1/api/pay/

Get information about your purchase.

#### Response

```
{
  "provider": "bango",
  "pay_url": "https://url.to-start.the/transaction"
}
```

##### Status Codes

- **200** – successfully completed.
- **400** – trans\_id is not set in the session.
- **404** – transaction could not be found.

## 1.2.4 Simulate

If a simulated payment is pending in the current session, as indicated by *the Pay API*, you can use this API to execute the simulated payment. This sends a server notice to the app that initiated the purchase so it can fulfill the simulated purchase.

**POST /mozpay/v1/api/simulate/**

Execute a pending simulated payment.

### Request

(no parameters)

### Response

(no parameters)

### Status Codes

- **204** – successful.
- **400** – invalid request.
- **403** – no pending simulation in the current session or invalid user permissions.

## 1.3 Solitude API Client

`lib.solitude.api.client`

An instantiated SolitudeAPI object using `settings.SOLITUDE_URL`

**class** `lib.solitude.api.BangoProvider` (*slumber*)

Bango payment provider

**class** `lib.solitude.api.BokuProvider` (*\*args, \*\*kw*)

The Boku payment provider.

**exception** `TransactionError`

Error relating to a Boku transaction.

**exception** `lib.solitude.api.BuyerNotConfigured`

The buyer has not yet been configured for the payment.

**class** `lib.solitude.api.PayProvider` (*slumber*)

Abstract payment provider

This encapsulates some API logic specific to payment providers such as configuring a new payment, creating products, etc.

**create\_product** (*generic\_product, provider\_seller, external\_id, product\_name*)

Creates and returns a provider-specific product object from Solitude.

**create\_transaction** (*generic\_buyer, generic\_seller, generic\_product, provider\_product, provider\_seller\_uuid, product\_name, transaction\_uuid, prices, user\_uuid, application\_size, source, icon\_url, mcc=None, mnc=None*)

Create a provider specific transaction and a generic Solitude transaction.

Return the provider a tuple of:

(transaction ID, payment start URL)

**get\_notification\_data** (*request*)

Given a provider-specific GET/POST request, return a dict of notification data that can be used for verification.

For example, a provider might notify Webpay of a successful transaction. That request might include a signature that can be used to verify authenticity.

**get\_product** (*generic\_seller, generic\_product*)

Returns the provider specific product object from Solitude.

**get\_seller** (*generic\_seller, provider\_seller\_uuid*)

Returns a provider-specific seller object from Solitude.

**transaction\_from\_notice** (*parsed\_qs*)

Get the Solitude transaction ID from the query string on a notification URL.

**verify\_notification** (*data*)

Verify provider notification using params from `get_notification_data()`.

This will raise an exception on any kind of verification error. This will also raise an exception if the transaction has already been processed.

Returns the Solitude transaction UUID.

**class** `lib.solitude.api.ProviderHelper` (*name, slumber=None*)

A common interface to all payment providers.

**create\_product** (*external\_id, product\_name, generic\_seller, provider\_seller\_uuid, generic\_product=None*)

Creates a generic product and provider product on the fly.

This is for scenarios like adhoc in-app payments where the system might be selling a product for the first time.

**server\_notification** (*request*)

Handles the server to server notification that is sent after a transaction is completed.

Returns the Solitude transaction UUID.

**start\_transaction** (*transaction\_uuid, generic\_seller\_uuid, provider\_seller\_uuid, product\_id, product\_name, prices, icon\_url, user\_uuid, application\_size, source='unknown', mcc=None, mnc=None*)

Start a payment provider transaction to begin the purchase flow.

**classmethod supported\_providers** (*mcc=None, mnc=None*)

Given the user's mobile network (when available) return a list of all suitable provider helper objects in order of preference.

Keyword arguments:

**mcc** The user's mobile carrier code, if known.

**mnc** The user's mobile network code, if known.

**class** `lib.solitude.api.ReferenceProvider` (*slumber*)

A reference payment provider

Our current reference implementation is known as Zippy.

This is our ideal API. If possible, other payment providers should follow this API.

If this provider is fully compliant it probably shouldn't need to override any of the inherited methods.

**exception** `lib.solitude.api.SellerNotConfigured`

The seller has not yet been configured for the payment.

`class lib.solitude.api.SolitudeAPI (*args, **kw)`

A Solitude facade that works with a payment provider or the generic Solitude API.

**Parameters** `url` – URL of the solitude endpoint.

`change_pin (uuid, pin, etag='', pin_confirmed=False, clear_was_locked=False)`

Changes the pin of a buyer, for use with buyers who exist without pins.

**Parameters**

- **integer** (*buyer\_id*) – ID of the buyer you'd like to change the PIN for.
- **pin** – PIN the user would like to change to.
- **pin\_confirmed** – Boolean to set if the PIN was already confirmed in the UI.
- **clear\_was\_locked** – Boolean to clear the `pin_was_locked_out` state if the PIN was changed by the user.

**Return type** dictionary

`confirm_pin (uuid, pin)`

Confirms the buyer's pin, marking it as confirmed in solitude

**Parameters**

- **uuid** – String to identify the buyer by.
- **pin** – PIN to confirm

**Return type** boolean

`create_buyer (uuid, email, pin=None, pin_confirmed=False)`

Creates a buyer with an optional PIN in solitude.

**Parameters**

- **uuid** – String to identify the buyer by.
- **pin** – Optional PIN that will be hashed.
- **pin\_confirmed** – Optional boolean to set if the PIN was already confirmed in the UI.

**Return type** dictionary

`get_active_product (public_id)`

Retrieves a an active seller product by its `public_id`.

**Parameters** **public\_id** – Product `public_id`.

**Return type** dictionary

`get_buyer (uuid, use_etags=True)`

Retrieves a buyer by their `uuid`.

**Parameters** **uuid** – String to identify the buyer by.

**Return type** dictionary

`reset_confirm_pin (uuid, pin)`

Confirms the buyer's pin, marking it as confirmed in solitude

**Parameters**

- **uuid** – String to identify the buyer by.
- **pin** – PIN to confirm

**Return type** boolean

**set\_new\_pin** (*uuid*, *new\_pin*, *etag*='')

Sets the `new_pin` for use with a buyer that is resetting their pin.

**Parameters**

- **integer** (*buyer\_id*) – ID of the buyer you'd like to change the PIN for.
- **pin** – PIN the user would like to change to.

**Return type** dictionary

**update\_buyer** (*uuid*, *etag*='', *\*\*kwargs*)

Updates a buyer identified by their `uuid`.

**Parameters** **uuid** – String to identify the buyer by.

**Return type** dictionary

**verify\_pin** (*uuid*, *pin*)

Checks the buyer's PIN against what is stored in solitude.

**Parameters**

- **uuid** – String to identify the buyer by.
- **pin** – PIN to check

**Return type** dictionary

## 1.4 Localization Testing

We are using a fake translation script that is mentioned on Ned Batchelder's blog called `poxx.py`. The specific version we are using was lifted from [Fjord](#).

What it does it is makes a translation for locale `xx` that turns all the strings into looking like something the [Swedish Chef](#) would say. There are some basic requirements for using it. You'll need to install `polib` like so:

```
pip install polib
```

As well as `gettext` for OSX:

```
brew install gettext
brew link gettext
```

Or Ubuntu:

```
apt-get install gettext gettext-tools
```

Once you have the requirements you can run the script with the command:

```
./bin/test_locales.sh
```

You'll need to tweak your `webpay/settings/local.py` with the setting:

```
LANGUAGE_CODE = 'xx'
```

Then you should be able to `./manage.py runserver` like normal and see everything translated. It should be very notable if the string is not translated. After updating your code/templates with your new translations you just simply run `locale_test.sh` again and it will regenerate the `xx` locale for you!

## 1.5 Services

Here are some web API services offered by WebPay. The production API domain is: <https://marketplace.firefox.com/>

### 1.5.1 Error Legend

When a user experiences a payment error triggered by your app, they see a message to help them figure out what to do. The error does not help you figure out what to do as the app developer. Instead the error contains a readable code at the bottom to indicate the cause of the error. You can use the legend API to get detailed info in your locale about what each error code means.

**GET /mozpay/services/error\_legend**

#### Request

##### Parameters

- **locale** – An optional language code for which to localize the legend. Example: `en-us` or `pl`. Take a look at our [PROD\\_LANGUAGES](#) setting for all possible codes.

#### Response

Example:

```
{
  "locale": "en-us",
  "errors": null,
  "legend": {
    "SOME_ERROR_CODE": "Detailed error explanation.",
    ...
  }
}
```

##### Status Codes

- **200** – success.
- **400** – request was invalid.

### 1.5.2 Signature Check

This API lets you validate an innocuous JWT with your issuer key and secret. This is used by the Firefox Marketplace as a system check to make sure all keys and secrets are configured correctly. It will return an error if the JWT issuer is unknown or if the signature is invalid. It's nicer to find this out from a system check rather than when a user is trying to purchase one of your products. Any app that is registered to [sell products via Firefox Marketplace](#) can use this API. For example, the Firefox Marketplace has a complimentary [signature check API](#) that can be used to generate a JWT for verification.

**POST /mozpay/services/sig\_check**

#### Request

##### Parameters

- **sig\_check\_jwt** (*string*) – a JWT issued by an app set up for payments. The typ must be correct. Example:

```
{ "iss": "YOUR_APP_ID",  
  "aud": "marketplace.firefox.com",  
  "typ": "mozilla/payments/sigcheck/v1",  
  "iat": timestamp(),  
  "exp": timestamp(),  
  "request": {} }
```

## Response

Example of a valid response:

```
{  
  "result": "ok",  
  "errors": {}  
}
```

Example of an invalid response:

```
{  
  "result": "error",  
  "errors": { "sig_check_jwt": ["INVALID_JWT_OR_UNKNOWN_ISSUER"] }  
}
```

## Parameters

- **result** (*string*) – either `ok` or `error`
- **errors** (*object*) – a map of validation errors that occurred for each input field

## Status Codes

- **200** – the JWT is valid.
- **400** – the JWT is invalid.

## 1.5.3 Exception Tester

You can use this endpoint to test how the application handles exceptions. When you make a GET request it will trigger an exception.

**GET** `/mozpay/services/exception/`

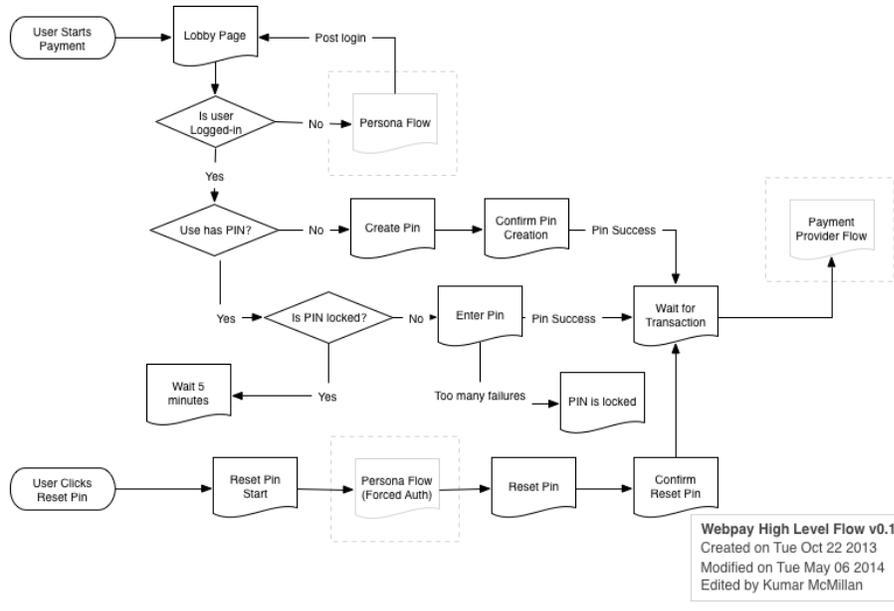
**Response**

### Status Codes

- **500** – internal error.

## 1.6 Flows

This document lists some diagrams detailing the flows through webpay.



---

## Indices and tables

---

- *genindex*
- *modindex*
- *search*



## /mozpay

GET /mozpay/services/error\_legend, 12  
GET /mozpay/services/exception/, 13  
POST /mozpay/services/sig\_check, 12  
GET /mozpay/v1/api/pay/, 7  
POST /mozpay/v1/api/pay/, 7  
GET /mozpay/v1/api/pin/, 5  
PATCH /mozpay/v1/api/pin/, 6  
POST /mozpay/v1/api/pin/, 6  
POST /mozpay/v1/api/pin/check/, 6  
POST /mozpay/v1/api/simulate/, 8



|

`lib.solitude.api`, 8



**B**

BangoProvider (class in lib.solitude.api), 8

BokuProvider (class in lib.solitude.api), 8

BokuProvider.TransactionError, 8

BuyerNotConfigured, 8

**C**

change\_pin() (lib.solitude.api.SolitudeAPI method), 10

client (in module lib.solitude.api), 8

confirm\_pin() (lib.solitude.api.SolitudeAPI method), 10

create\_buyer() (lib.solitude.api.SolitudeAPI method), 10

create\_product() (lib.solitude.api.PayProvider method), 8

create\_product() (lib.solitude.api.ProviderHelper method), 9

create\_transaction() (lib.solitude.api.PayProvider method), 8

**G**

get\_active\_product() (lib.solitude.api.SolitudeAPI method), 10

get\_buyer() (lib.solitude.api.SolitudeAPI method), 10

get\_notification\_data() (lib.solitude.api.PayProvider method), 8

get\_product() (lib.solitude.api.PayProvider method), 9

get\_seller() (lib.solitude.api.PayProvider method), 9

**L**

lib.solitude.api (module), 8

**P**

PayProvider (class in lib.solitude.api), 8

ProviderHelper (class in lib.solitude.api), 9

**R**

ReferenceProvider (class in lib.solitude.api), 9

reset\_confirm\_pin() (lib.solitude.api.SolitudeAPI method), 10

**S**

SellerNotConfigured, 9

server\_notification() (lib.solitude.api.ProviderHelper method), 9

set\_new\_pin() (lib.solitude.api.SolitudeAPI method), 10

SolitudeAPI (class in lib.solitude.api), 9

start\_transaction() (lib.solitude.api.ProviderHelper method), 9

supported\_providers() (lib.solitude.api.ProviderHelper class method), 9

**T**

transaction\_from\_notice() (lib.solitude.api.PayProvider method), 9

**U**

update\_buyer() (lib.solitude.api.SolitudeAPI method), 11

**V**

verify\_notification() (lib.solitude.api.PayProvider method), 9

verify\_pin() (lib.solitude.api.SolitudeAPI method), 11