WireXfers Documentation

Release 2014.06-dev

Priit Laes

June 21, 2016

Contents

Release v2014.06-dev.

WireTransfers is an *ISC Licensed* online payments library, written in Python, supporting various online payment protocols (IPizza, Solo/TUPAS) using a simple API.

CHAPTER 1

User Guide

This part of documentation begins with some background information, then focuses on step-by-step instructions for making online payments using WireTransfers library.

1.1 Introduction

1.1.1 Philosophy

WireTransfers is being developed while keeping a few **PEP 20** idioms in mind:

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Readability counts.

Therefore all contributions to WireTransfers should keep these important words in mind.

1.1.2 ISC License

WireTransfers is released under the terms of The ISC License.

"Why the ISC license?", you may ask? That's because this license allows software to be used freely in proprietary, closed-source software.

1.1.3 WireTransfers License

CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

1.2 Using WireXfers

This section should give you an introduction on how to integreate WireXfers with various applications and web frameworks.

Note: Code snippets below use pseudocode and will not work when using them in real application. Consult framework-specific examples for real working code.

Basic flow of the payment process is following:

1. Initialize provider-specific keychain
2. Initialize provider
3. Create payment information
4. Show user the payment form which takes him to provider page
5. Process the return results

1.2.1 Setting up the provider

Each provider has to be initialized by provider-specific keychain. Depending on keychain, its arguments are either simple strings (Solo/TUPAS) or consists of private/public key pair objects (IPizza).

```
from wirexfers.providers import PseudoProvider

# Create and initialize provider-specific keychain
keychain = PseudoProvider.KeyChain(...)

# Create and initialize the provider
#: user - user id used at provider's side
#: endpoint - endpoint address where to send payment request
provider = PseudoProvider(user, keychain, endpoint)
```

1.2.2 Creating the payment and initializing the payment request

In order to make a payment, we first need to set up a payment information by filling out relevant fields of *PaymentInfo*.

```
from wirexfers import PaymentInfo, utils
info = PaymentInfo('1.00', 'Test transfer', utils.ref_731('123'))
```

Now that we have the *PaymentInfo*, in order to create a payment request, we have to create a dictionary containing return urls to views where our application handles the payment response.

Note: Return url support varies with providers. Please consult each providers documentation to see which return urls are supported. By default we need at least the `return` URL.

Note: Return urls should be absolute!

```
urls = {'return': 'http://example.com'}
```

With that, everything we need to create a payment request (*PaymentRequest*) has been done:

```
payment = provider(info, urls)
```

All we need now is to pass the `payment` info into template and create a HTML form visible to the user:

This is all from application side, we just have to pass the `payment` to the template in order to show the payment form to the user. Lets assume that payment request has been passed into template context as `payment` variable, so can use *form* iterator to create form fields, *info* to display the payment information and various *provider* fields to initialize a simple HTML form.

Basic Jinja2 template should look like this:

```
<form method="POST" action="{{ payment.provider.endpoint }}" accept-charset="{{ payment.provider.form
{% for item in payment.form -%}
    {% set name, value = item -%}
    <input name="{{ name }}" value="{{ value }}" type="hidden">
{% endfor -%}
<dl>
  <dt>Amount:</dt>
  <dd>{{ payment.info.amount }}</dd>
  <dt>Message:</dt>
  <dd>{{ payment.info.message }}</dd>
</dl>
<input type="submit">
</form>
```

1.2.3 Handling the Payment response

Note: Depending on the provider, we need either handle single or multiple return urls.

Note: Depending on the provider we need to either handle GET or POST request data.

Note: Depending on the provider we also need to handle responses in non-utf8 charsets.

In order to verify payment status, we just need to parse the request data using *parse_response()*. This create a *PaymentResponse* which contains `is_valid` and various other data related to payment.

```
from wirexfers.exc import InvalidResponseError
# data contains either POST or GET request data
try:
    payment = provider.parser_response(data)
except InvalidResponseError
    # Signature failure, we should redirect to proper error page
    pass

if payment.is_valid:
```

```
    # Show "Successful order page!"
else:
    # Show "Order failure page"
```

And that's basically how it works! :)

1.3 Supported Payment Providers

List of currently supported protocols:

- IPizza
- Solo/TUPAS

1.3.1 Providers supporting IPizza protocol

- Supported Estonian banks:
  - Danske Bank Estonia - *wirexfers.providers.ipizza.EEDanskeProvider*
  - Krediidipank - *wirexfers.providers.ipizza.EEKrediidipankProvider*
  - LHV Bank Estonia - *wirexfers.providers.ipizza.EELHVProvider*
  - SEB Bank Estonia - *wirexfers.providers.ipizza.EESEBProvider*
  - Swedbank Estonia - *wirexfers.providers.ipizza.EESwedBankProvider*

1.3.2 Providers supporting Solo/TUPAS protocol

- Supported Estonian banks:
  - Nordea Estonia - *wirexfers.providers.tupas.EENordeaProvider*

1.4 Generating private-public keypair

Generating the private RSA key with 4096-bit keysize:

```
$ openssl genrsa -out privkey.pem 4096
```

Generate the Certificate Request:

```
$ openssl req -new -key privkey.pem -out certificate-request.csr
```

CHAPTER 2

Integrating with frameworks

Integration examples with various frameworks:

- Flask - https://github.com/plaes/wirexfers-flask-demo

CHAPTER 3

API Documentation

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

3.1 API

This part of the documentation covers all the interfaces of WireXfers.

3.1.1 Payment Providers

IPizza

class wirexfers.providers.ipizza.**IPizzaProviderBase**(*user*, *keychain*, *endpoint*, *extra_info={}*)

Base class for IPizza protocol provider.

Protocol IPizza

KeyChain IPizzaKeyChain

Supported return urls:

- return

Supported protocol version:

- 008

parse_response(*form*, *success=True*)
Parse and return payment response.

IPizza Providers

class wirexfers.providers.ipizza.**EEDanskeProvider**(*user*, *keychain*, *endpoint*, *extra_info={}*)

Danske Bank A/S Eesti filiaal
http://www.danskebank.ee

Protocol IPizza

KeyChain KeyChain

Supported return urls:

- return

Supported protocol version:

- 008

parse_response(*form*, *success=True*)
Parse and return payment response.

class wirexfers.providers.ipizza.**EEKrediidipankProvider**(*user*, *keychain*, *endpoint*, *extra_info={}*)

AS Eesti Krediidipank
http://krediidipank.ee/

Protocol IPizza

KeyChain KeyChain

Supported return urls:

- return

Supported protocol version:

- 008

parse_response(*form*, *success=True*)
Parse and return payment response.

class wirexfers.providers.ipizza.**EELHVProvider**(*user*, *keychain*, *endpoint*, *extra_info={}*)

AS LHV Pank
https://www.lhv.ee

Protocol IPizza

KeyChain KeyChain

Supported return urls:

- return

Supported protocol version:

- 008

parse_response(*form*, *success=True*)
Parse and return payment response.

class wirexfers.providers.ipizza.**EESEBProvider**(*user*, *keychain*, *endpoint*, *extra_info={}*)

AS SEB Pank
http://www.seb.ee

Protocol IPizza

KeyChain KeyChain

Supported return urls:

- `return`

Supported protocol version:

- `008`

parse_response(*form*, *success=True*)
Parse and return payment response.

class `wirexfers.providers.ipizza.`**EESwedBankProvider**(*user*, *keychain*, *endpoint*, *extra_info={}*)

SWEDBANK AS
https://www.swedbank.ee

Protocol IPizza

KeyChain `KeyChain`

Supported return urls:

- `return`

Supported protocol version:

- `008`

parse_response(*form*, *success=True*)
Parse and return payment response.

Solo/TUPAS

Solo/TUPAS providers

class `wirexfers.providers.tupas.`**EENordeaProvider**(*user*, *keychain*, *endpoint*, *extra_info={}*)

Nordea Bank Finland Plc Eesti / AS Nordea Finance Estonia
https://www.nordea.ee

Protocol Solo/TUPAS

KeyChain `KeyChain`

Supported return urls:

- `cancel` - user cancels payment
- `reject` - bank rejects payment (due to insufficient funds, ...)
- `return` - payment is successful

Supported protocol version:

- `0003`

parse_response(*form*, *success=True*)
Parse and return payment response.

3.1.2 Base Classes

class wirexfers.providers.**KeyChainBase**
Base class for protocol-specific key handling.

class wirexfers.providers.**ProviderBase**(*user*, *keychain*, *endpoint*, *extra_info={}*)
Base class for all payment providers.

endpoint = None
Endpoint address used to initiate payment requests.

extra_info = None
Dictionary containing extra user-supplied information. Can be used for supplying provider url, etc.

keychain = None
Protocol-specific keychain implementation - *wirexfers.providers.KeyChainBase*

parse_response(*data*)
Parse the payment request.

Parameters **form** – Raw payment response data.

user = None
User id for payment processor.

ProviderBase.**__call__**(*payment*, *return_urls*)
Create and return a payment request.

Parameters **payment** (*PaymentInfo*) – payment information

Return type *PaymentRequest*

class wirexfers.**PaymentRequest**(*provider*, *info*, *return_urls*)
PaymentRequest class.

Parameters

- **provider** (*ProviderBase*.) – Payment provider
- **info** (*PaymentInfo*.) – Payment information
- **return_urls** (Dict) – Dictionary of return URLs. Depends on the specific provider, but generally `{'return': ... }` is required.

Raises ValueError when invalid configuration is detected.

form = None
List containing `(name, value)` tuples for HTML-form setup.

info = None
PaymentInfo containing various payment information (sum, etc..)

provider = None
ProviderBase that handles the payment request.

class wirexfers.**PaymentResponse**(*provider*, *data*, *successful=False*)
PaymentResponse class.

data = None
Dictionary containing payment-related data, specific to provider

provider = None
ProviderBase that handles the payment request.

successful = None
Whether payment response is successful (some providers don't provide this status, therefore allow setting it from the view)

3.1.3 Exceptions

wirexfers.exc

Exceptions used with WireXfers.

The base exception class is *WireXfersError*

copyright

3. 2012-2014 Priit Laes

license ISC, see LICENSE for more details.

exception wirexfers.exc.**InvalidResponseError**
Bases: *wirexfers.exc.WireXfersError*

Raised when an invalid payment response data is supplied to the response parser.

exception wirexfers.exc.**WireXfersError**
Bases: exceptions.Exception

Generic error class.

3.1.4 Utility Classes

class wirexfers.**PaymentInfo**(*amount*, *message*, *refnum*)
Payment information required for *PaymentRequest*.

amount = None
Payment amount as string, uses . as decimal point separator.

message = None
Message used for payment description.

refnum = None
Reference number.

3.1.5 Utility Functions

wirexfers.utils

This module provides utility functions that are used within WireXfers, but might be also useful externally.

copyright

3. 2012-2014 Priit Laes

license ISC, see LICENSE for more details.

wirexfers.utils.**load_key**(*path*, *password=None*)
Import an RSA key (private or public half).

Parameters

- **path** (*string*) – path to key half.
- **password** (*string or None*) – password for private key.

Return type Crypto.PublicKey.RSA._RSAobj

wirexfers.utils.**ref_731**(*n*)

Reference number calculator. Returns reference number calculated using 7-3-1 algorithm used in Estonian banks.

Parameters **n** (*string*) – base number (client id, etc)

Return type string

W

Symbols

A

D

E

F

I

K

L

M

P

R

S

U

W