Quick Instructions:

Quick instructions updated 27.4.2012.
Currently confirmed working operations as of 27.4.2012 against Osuuspankki web service are certificate request and transaction query.
For the transaction-query, Osuuspankki supports only requesting of the current date’s transactions, so it would be advisable to perform transaction-query against web service -channel close to midnight.

In case of technical error from the bank, call +358 100 05151 if you’re using Osuuspankki.

Software requirements:

Before starting anything else, the following software and libraries need to be installed.

- Python 3.1 ->	
- OpenSSL	
- Suds	(https://bitbucket.org/bernh/suds-python-3-patches/wiki/Home The easies way is to download suds_patched.zip directly and install from it)
- PyOpenSSL 0.13	(http://pypi.python.org/pypi/pyOpenSSL/0.13 You need to install python header files if you install it from source)
- LXML	(http://lxml.de/index.html#download or apt-get install python3-lxml in Debian and Ubuntu or python3-lxml in Fedora (->13))
- PyCrypto	(http://www.pycrypto.org/ , http://pypi.python.org/pypi/pycrypto/2.5 or apt-get install python3-crypto in Debian (testing, unstable) and Ubuntu (precise) or python3-crypto in Fedora (->17))

Other requirements:

- CustomerID (Käyttäjätunnus) 10 characters
- Transfer key (siirtoavain) 16 characters

Which are given by the bank when user makes deal with them (or actually user gets only first 8 characters for transfer key, rest is send by SMS or letter)
NOTE: With Osuuspankki, your transfer key will expire in 3 months after issuing!

- 2048 bit RSA key
- PKCS#10 formatted certificate request.

Which need to be generated by user (Next chapter tells how you can do it with OpenSSL)

Generating key and certificate request with OpenSSL-software:

```
user@host:~$ openssl req -newkey rsa:2048 -keyout key.pem -out req.der -outform DER
```

Password will be removed later so just put something simple on it.
After key has been generated OpenSSL will ask some information.

- Put FI in country field.
- Put your CustomerID to Common Name
- Rest of the fields can be nulled with '.', even the challenge password.

Remove pass phrase from private key as PyCrypto doesn't support it

```
user@host:~$ cp key.pem  key.pem.org
user@host:~$ openssl rsa -in key.pem.org -out key.pem
```
(will ask the password you previously gave)

Security notice: Keep your private key and certificate safe. If you suspect that your private key has fall into the wrong hands contact your bank immediately so they can put your certificate to revocation list. If you need to revoke your certificate, call 010 252 8470 (Osuuspankki)

Step-by-step instructions:

These can be inserted directly in python interpreter or preferably in separate python source-file.

Certificate related stuff:

If you don't have certificate or your old certificate isn't valid any more.

0. Import required modules.
```
>>> from bankws.idservice import OPCertificateRequest
```
1. Create OPCertificateRequest object depending on environment.
```
# OPCertificateRequest(CustomerID, Environment)
>>> opc = OPCertificateRequest("1000000000", "TEST")
or
>>> opc = OPCertificateRequest("1000000000", "PRODUCTION")
```
2. Use request_with_transferkey to get first certificate (certificate will be written to file)
```
# opc.request_with_transferkey(Transferkey, Certificate request filename, certificate
# filename)
>>> opc.request_with_transferkey("1234567890123456", "req.der", "op.crt")
```

NOTE: At least currently(spring 2012) Osuuspankki returns 26 technical error if your transferkey has expired.

If you have certificate:

0. Import needed modules.
```
>> from bankws.idservice import OPCertificateRequest
>> from bankws.certificate import check_renewable_status
```
1. Check certificate status with check_renewable_status
```
# check_renewable_status(certificate filename)
>> check_renewable_status("op.crt")
```

1.1. Check_renewable_status returns False:
   - You don’t need to do anything

1.2 Check_renawable_status returns True:
   - You need to generate new Private key and Certificate Request
   - Get them signed by sending request_with_old_certificate

```
# OPCertificateRequest(CustomerID, Environment)
>>> opc = OPCertificateRequest(“1000000000”, “TEST”)
# request_with_old_certificate(RSA-key, Certificate request, old certificate, new
# certificate filename)
>>> opc.request_with_old_certificate(“priv_key.pem”, “newreq.der”, “op.crt”, “new.crt”)
```

1.3 Check_renewable_status raises exception:
   - Certificate is not valid anymore, you need to contact bank to get new transfer key.

Make Transaction query (Tapahtumaotekysely):

0. Import required modules

```
>> from bankws.webservice import WebService
>> from bankws.bank import OP
```

(1. Check certificate status)

2. Create WebService object.

```
#   Ws = WebService(CustomerID, RSA - key, Certificate, Bank, environment)
>>> ws = WebService(“1000000000”, “privkey.pem”, “op.crt”, OP, “TEST”)
or
>>> ws = WebService(“1000000000”, “privkey.pem”, “op.crt”, OP, “PRODUCTION”)
```

3. Call transaction_query(account_number) method.

```
>>> TLR = ws.transaction_query(“112233-44556677”)
or
>>> TLR = ws.transaction_query(“FI1122334455667788”)
```

4. Check did the response contain information record. (If response contains information record it means that bank had some problem in the request processing.)

```
>>> result = TLR.is_problem_free()
```

5. Get values that you need from returned object.

```
# If result is True
>>> for transaction in TLR.transactions:
                print(transaction.name)                      # Payer/Payee name
                print(transaction.reference_number)          # Reference number
                print(transaction.money)             # Amount of money moved
# If Result is False
>>> for message in TLR.information_record.messages:
                print(message)
```

Create C2B-request:

To create the C2B-request (a request to make a payment) to be sent to the bank, there are a lot of different variables needed for the message. At the end of this document, the same steps are illustrated with an example values (though, for security reasons, still without the identification information for particular customer)

The steps for creating the payment:

1. Import the required modules:

```
>>> from bankws import c2b
>>> from bankws import c2bhelper
>>> from bankws import entity
>>> from bankws import webservice
>>> from bankws import bank
```

2. Create the entities used in the message:

```
>>> company = entity.Entity(CompanyName, CompanyCountry, CompanyAddressLine, CompanyAddressLine,
CompanyAddressLine)
```

company: Object holding debtor’s information
(string) **CompanyName**: Company’s name who originates the payment
(string) **CompanyCountry**: Company’s country of residence
(string) **CompanyAddressLine**: Company’s address line, e.g. “Teollisuuskatu 2” or “90800”. Maximum of 5 address lines are supported

```
>>> receiver = entity.Entity(CreditorName, CreditorCountry, CreditorAddressLine, CreditorAddressLine,
CreditorAddressLine)
```

receiver: Object holding creditor’s information
(string) **CreditorName**: Creditor’s name
(string) **CreditorCountry**: Creditor’s country
(string) **CreditorAddressLine**: Creditor’s address line, e.g. “Kotikatu 1” or “10231” or “OULU”. Maximum of 5 address lines are supported

3. Create the payment-message and add the group header for the payment

```
>>> payment = c2b.C2B(company)
>>> payment.group_header(MessageID, Grpg)
```

company: Object holding debtor’s information
payment: Object holding the information to be sent to bank
(string) **MessageID**: Message ID, unique for atleast 3 months
(string) **Grpg**: Use value “MIXD” if there is one or more transactions in one payment

4. Create the SEPA-payment

```
>>> sepa = c2bhelper.SEPA(company, PaymentMethod, ServiceLevel, ReqdExctnDt,
                    MaterialId, DbtrIBAN, DbtrBIC, ChrgBr, PmtInfId, InstrPrty,
                    CCY, UltmtDbtr)
```

sepa: Object holding payment’s information
(string) **PaymentMethod**: Use value “TRF” in SEPA-payments
(string) **ServiceLevel**: Use “SEPA” in service level code
(string) **ReqdExctnDt**: Requested expiration date for the payment in form “YYYY-MM-DD”

(string) **MaterialID**: Value is “Maksatustunnus” from debtor’s C2B-contract
(string) **DbtrIBAN**: Debtor’s bank account number in IBAN-format
(string) **DbtrBIC**: Debtor’s BIC-code
(string) **ChrgBr**: “Kulukoodi”, use “SLEV” in SEPA-payments
(string) **PmtInfId**: Identification for the instalment (seen only by the debtor). Use “None” if nothing else if given
(string) **InstrPrty**: Instruction priority, available values for this are “HIGH” or “NORM”. Use “None” and priority is read from the transaction.
(string) **CCY**: Currency for the payment, the use of “EUR” is recommended. Use “None” if the currency is unspecified.
(string) **UltmtDbtr**: Ultimate debtor for this payment. Use “None” if nothing else is specified

5. Create a transaction and add it to the list of transactions in the SEPA-payment

```
>>> tr = c2bhelper.CdtTrfTxInf(receiver, PmtPurp, EndToEndId, InstdAmt, InstdAmtCurr, CdtrBIC,
                        CdtrAcct, ChrgBr, InstrId, InstrPrty, SosSecNbr,
                                   UstrdMsg, UltmtCdtr)
>>> sepa.add_transaction(tr)
```

receiver: Object holding creditor’s information
tr: Object holding transaction’s information
sepa: Object holding payment’s information
(string) **PmtPurp**: One of [STDY, BECH, PENS, BENE, SSBE, AGRT, SALA, TAXS] or use “None” if none of aforesaid are eligible
(string) **EndToEndId**: ID used to uniquely identify the transaction
(float) **InstdAmt**: The amount for the transaction
(string) **InstrdAmtCurr**: The currency for the transaction
(string) **CdtrBIC**: Creditor’s BIC
(string) **CdtrAcct**: Creditor’s account number in IBAN-format
(string) **ChrgBr**: “Kulukoodi”, use “SLEV” in SEPA-payments
(string) **InstrId**: Transaction ID, seen on debtor’s account statement. Use “None” if no ID is given
(string) **InstrPrty**: Transaction priority, values are “HIGH” or “NORM”. If not given, use “None”
(string) **SosSecNbr**: Social security number of the creditor, if not known, use “None”
(string) **UstrdMsg**: Free-text message for the creditor, use “None” if nothing is specified
(string) **UltmtCdtr**: Ultimate creditor, use “None” if not eligible

6. Add the SEPA-payment to the message

```
>>> payment.add_SEPA_payment(sepa)
```

sepa: Object holding payment’s information
payment: Object holding the information to be sent to bank

7. Create the XML-file in string-format and put it in a variable

```
>>> payment_xml = str(payment)
```

Repeat steps 4 and 5 as necessary. After these seven (7) steps, the message is ready to be handed over to WebService's upload_file()-method to be sent to the bank.

Debugging/Logging:

If you ran into problems or are just curious about what's happening you can increase verbosity level of logger.

```
>>> import logging
>>> logging.basicConfig(level=logging.INFO)
>>> logging.getLogger("bankws").setLevel(logging.DEBUG)
```

If that is not enough you can try:

```
>>> logging.basicConfig(level=logging.DEBUG)
```

Quick reference:

Checks does certificate need renewal:

```
>>> check_renewable_status(certificate_filename)
```

Arguments: certificate filename as a string
Return value: Boolean (is the certificate renewable or not)
Exceptions: RuntimeError (Certificate has expired.)
EnvironmentError (filename points to non-existing file)

Get new certificate with transfer key:

```
>>> op = OPCertificateRequest(CustomerID, environment)
```

Arguments: CustomerID as a string (or int)
environment as a string (either TEST or PRODUCTION)
Exceptions: ValueError (environment doesn't contain supported value)

```
>>> op.request_with_transferkey(Transferkey, Certificate request filename, filename for new
                                 certificate)
```

Arguments: Transfer key as a number sequence
Certificate request as filename that points to certificate request.
Filename for the returned certificate.
Return value: Boolean telling was the request successful

Renew certificate:

```
>>> op = OPCertificateRequest(CustomerID, environment)
>>> op.request_with_old_certificate(RSA-key filename, Certificate request filename,
                                    old certificate filename, new certificate filename)
```

Arguments: RSA-key as a filename that points to the RSA-key.
Certificate request as filename that points to certificate request.
Old certificate as filename that points to old certificate.
Filename to where new certificate will be saved.
Return value: Boolean telling was the request successful.

Send (XML) data to bank:

```
>>> ws = WebService(CustomerID, RSA - key filename, Certificate filename, Bank, environment)
```

Arguments: CustomerID as a string
RSA - key as a filename that points to the RSA-key
Certificate as a filename that points to the signed certificate
Bank as a Bank object that holds webservice addresses and BIC number.
Environment as string (TEST or PRODUCTION, defaults TEST)

Exceptions: ValueError (If Environment contains unsupported value)
ValueError (RSA-key or Certificate points to non-existing file.)
ValueError (Bank object is missing needed fields.)

```
>>> ar = ws.upload_file(content, filetype)
```

Arguments: Content as a string (Data that you want upload to bank)
filetype of uploaded data as a string. (default value is pain.001.001.02 which is meant for C2B data)

Return value: ApplicationResponse object.

Exceptions: RuntimeError (If the upload file request wasn't accepted by the bank)

```
# You should save at least reference number to somewhere
>>> print(ar.references[0].reference) # Prints uploaded file reference number
```

Get file from bank:

```
>>> ws = WebService(CustomerID, RSA - key filename, Certificate filename, Bank object)
>>> ar = ws.download_file(File reference number)
```

Arguments: File reference number as a string. (Use for example downloadfilelist to determine correct reference number)

Return value: ApplicationResponse object.

Exceptions: RuntimeError (If the download file request wasn't accepted by the bank)

```
# Downloaded data is in the content field of returned object.
>>> print(ar.content)
```

List files saved on the bank:

```
>>> ws = WebService(CustomerID, RSA - key, Certificate, Bank object)
>>> ar = ws.download_filelist(File status)
```

Arguments: File status as a string (NEW, ALL, DLD) (new, all, downloaded)

Return value: ApplicationResponse object.

Exceptions: RuntimeError (if the download filelist request wasn't accepted by the bank)

```
# List of filerefences is in references field of returned object
>>> for reference in ar.references:
        print(reference) # prints all data saved on reference object
```

Transaction query (Tapahtumaotekysely™):

```
>>> ws = WebService(CustomerID, RSA - key, Certificate, Bank object)
>>> ret_val = ws.transaction_query(account_number)
```

Arguments: Account number as a string in BBAN or IBAN format
BBAN account number format: 112233-44556677
IBAN account number FI1122334455667788

Return value: TransactionListResponse

Exceptions: ValueError (if account number is not valid)
RuntimeError (if request wasn't accepted.)

Objects

Bank

Property	Description
BIC	Bank Identification Code
wsdl_url	Location of bank WebService WSDL file (Production)
wsdl_test_url	Location of bank WebService WSDL file (Test)
id_url	Location of bank identification service WSDL file. (Production)
id_test_url	Location of bank Identification service WSDL file. (Test)

Table 1. Bank object.

ApplicationResponse

Property	Description
content	Contains returned data. (In case of download file else this contains None)
references	Contains list of file references

Table 2. ApplicationResponse Interface.

TransactionListResponse

TransactionListResponse is object that contains all data parsed from message.

Property	Description
basic_record	Holds transaction basic record.
transactions	Holds list of Transaction objects.
balance_record	Holds balance record.
information_record	Holds information record (This is only generated if there were some error.)

Table 3. TransactionListResponse Interface.

Property	Description
transaction	Contains TransactionBasicRecord
extras	Contains list of TransactionExtraRecords.

Table 4. Transaction Interface

Property	Description
account	User account number
query_date	Date of the query
generation_time	When the list of transactions were generated.

account_information	Name, owner, limit and currency used.
bank	Name of the bank.

Table 5. BasicRecord interface.

Property	**Description**
transaction_marker	Transaction operation (Deposit, Intake, Fix for deposit, Fix for intake.)
description	Code and description of transaction. (Format “code: description”)
money	Amount of money in transaction.
name	Name of Payee/Payer
account	Payee account number (contains * account number has changed)
reference_number	Transaction reference number.

Table 6. TransactionBasicRecord interface.

Property	**Description**
content	Contains tuple(extra_record_type, textual_description).

Table 7. TransactionExtraRecord interface.

Property	**Description**
query_date	Date of the query.
balance	Account balance
available_balance	Amount of usable balance.

Table 8. Transaction BalanceRecord interface.

Property	**Content**
messages	List of messages returned from bank.

Table 9. Transaction InformationRecord interface.

FAQ:

Q: In key generation example key is named as privkey.pem even though part of command says outform DER. Shouldn’t the private key be named as privkey.der ?

A: No. OpenSSL syntax is little confusing here. Generated key will be in PEM - format. Certificate request and certificate that bank will return will be in DER - format.

Q: How long certificate is valid ?
A: Certificate will be valid 2 years. (At least according to the bank documentation, the test certificate that bank returned was valid only for 1 year.)

Q: When i need to renew my certificate ?
A: You can renew your certificate when it has less than 60 days valid time left. You can use check_renewable_status to check if certificate is renewable. (Test environment let's you renew your certificate whenever you want.)

Q: Does library make any checks to transferkey ?
A: Library makes Luhn modulus 10 check to transfer key which should detect typing errors in transfer key.

Q: Luhn's modulus 10 ?????
A: Luhn's modulus 10 is a simple algorithm that is used to calculate check marks for number series. It's used for example in credit card numbers and IMEI numbers. Read more from wikipedia http://en.wikipedia.org/wiki/Luhn_algorithm

Q: Library raises InvalidCertificateException. What to do ?
A: Secure transport relies on root certificate list which can be found in resources/cacert.pem file. If bank changes the CA authority it may be value that wasn't on that list. You can try fixing the problem with downloading new version of root certificate list from http://curl.haxx.se/ca/cacert.pem (Curl uses same list (which is originally converted from mozilla's certificate list).

Q: What is this Tapahtumaotekysely ?
A: Tapahtumaotekysely is a name for the service that returns list of that day transactions.

Q: I'm trying to use this library with some other bank than Osuuspankki. What is the BIC that is part of BANK object ?
A: BIC means Bank Identifier Code. You can check the code from http://www.fkl.fi/teemasivut/sepa/tekninen_dokumentaatio/Sivut/default.aspx -> Suomalaiset rahalaitostunnukset ja BIC-koodit (PDF file)

Q: I have read the source code. Why have you overrided SUDS default HTTP 500 Internal Service Error handler?
A: At least Osuuspankki returns rejected responses with 500 Internal Service Error. Suds default 500 Internal service handler crashes when it tries to parse that. (Suds think that response in that case contains only <name> and <reason> tags.)

Q: My BBAN account number doesn't contain 8 number after '-' ?
A: Don't worry, just enter your account number as it is. Library will take care of adding zero's to the account number.

Example code:

Notes:

CustomerID found on example is used in Osuuspankki examples so hopefully it doesn’t belong to anybody.
Transferkey used in example code isn’t valid luhn10 sequence (just to make sure that example doesn’t contain values that belong to somebody)

```python
''' Demonstration for BANKWS project.'''

import os

from bankws.idservice import OPCertificateRequest
from bankws import certificate
from bankws.webservice import WebService
from bankws.transactionlistresponse import TransactionListResponse
from bankws.bank import OP
from bankws import c2b_test


def main():
    # First check certificate related stuff
    print("Checking for existing certificate")
    if os.path.isfile("resources/op.crt"):
        print('Found existing certificate')
        try:
            status = certificate.check_renewable_status("resources/op.crt")
        except EnvironmentError:
            print('Unable to open certificate file.')
            return
        except RuntimeError:
            print("Certificate outdated.")
            print("Contact your bank to get new transferkey.")
            return
        if status:
            # Certificate has less than 60 days time left.
            print("You can renew your certificate")
            answer = input("Renew certificate (y,n)>>")
            if answer.lower() == 'y':
                op = OPCertificateRequest("1000000000",  # CustomerId
                                          "TEST")      # Environment
                if op.request_with_old_certificate(
                        "resources/key.pem",  # RSA - key
                        "resources/req.der",  # Certificate Request
                        "resources/op.crt",   # Old certificate
                        "resources/new.crt"): # Filename for new certificate
                    print("Request accepted")
                else:
                    print("There was problem in request processing.")
        else:
            print("Certificate OK.")
            print("Just for demonstration purpose print all certificates")
            op = OPCertificateRequest("1000000000",  # CustomerId
                                      "TEST")      # Environment

            status, response = op.get_certificates()
            if status:
                print("Prints common name from first certificate")
                # You can access certificates after the request so
                # you can for example save missing certificate if needed.
                print(response.certificates[0].name)
            else:
                print("There was problem in request processing.")
```

```
else:
    print("No existing certificate.")
    print("Getting new certificate.")
    answer = input("Ok(y/n)>>")
    if answer == "y":
        op = OPCertificateRequest("1000000000", "TEST")
        if op.request_with_transferkey(
                        "1234567890123456",     # transferkey
                        "resources/req.der",  # Certificate request
                        "resources/op.crt"    # Filename for new certificate
                        ):
            print("Saved certificate to resources/op.crt")
        else:
            print("There was problem in request processing.")

# Do some operation on webservice side
try:
    ws = WebService("1000000000",         # CustomerID
                    "resources/key.pem",  # Private RSA - key
                    "resources/op.crt",   # Signed Certificate
                    OP,                   # Bank object
                    environment="TEST")   # Environment
except ValueError as e:
    print(e)
    return

while True:
    print("Operations:")
    print("1.Upload File\n2.Download file\n3.Download filelist\n"
          "4.Make transaction query\n")

    selection = input("Select operation[1-4]")

    if selection == '1':
        print("Upload file")
        print("Upload c2b message to bank")
        content = c2b_test.pmt_xml
        try:
            ret_val = ws.upload_file(
                                content,           # Content to be uploaded
                                "pain.001.001.02"  # Filetype
                                )
        except RuntimeError as e:
            print(e)
        else:
            # Print out reference number for uploaded file so you can
            # use it the find out which file to download from bank.
            print(ret_val.references[0].reference)

    elif selection == '2':
        print('Download file')
        print("Use Download filelist to get list of reference numbers")
        refnumber = input("Reference number(-1 to cancel)>>")
        if refnumber == '-1':
            break
        try:
            # Downloads file from bank.
            ret_val = ws.download_file(
                                refnumber  # Refnumber is unique id for file
                                )
        except RuntimeError as e:
                print(e)
            else:
                print(ret_val.content)

        elif selection == '3':
            print('Download filelist')
            try:
                # Gets list of files that haven't been downloaded from bank.
                ret_val = ws.download_filelist('NEW')  # Supports NEW, DLD, ALL
            except RuntimeError as e:
                print(e)
            else:
                # If you have uploaded file to bank and bank has processed
                # file some of parentfile reference should contain the
                # referencenumber that bank returned in uploadfile operation.
                for reference in ret_val.references:
                    print(reference) # Prints all values saved on reference object
        elif selection == '4':
            print("Make transaction query to bank")
            print("Input your account number (in old format or IBAN number)")
            account = input("Account number>> ")
            try:
                ret_val = ws.transaction_query(account)
            except (RuntimeError, ValueError) as e:
                print(e)
            else:
                if isinstance(ret_val, TransactionListResponse):
                    if ret_val.is_problem_free():
                        for transaction in ret_val.transactions:
                            # Transaction type
                            print(transaction.transaction.description)
                            # Amount of money transferred.
                            print(transaction.transaction.money)
                            # Name of payee/payer
                            print(transaction.transaction.name)
                            # Payee account number
                            print(transaction.transaction.account)
                            # Reference number
                            print(transaction.transaction.reference_number)
                    else:
                        # If TLR wasn’t problem free it contains information record which
                        # contains some reason for the problem.
                        for message in ret_val.information_record.messages:
                            print(message)
                else:
                    # transaction_query returned ApplicationResponse
                    # TransactionList must be separately downloaded.
                    print(ret_val.references[0])
        else:
            print("Exit..")
            return

if __name__ == '__main__':
    main()
```

To create the payment-file and upload it to the bank, user can use the following lines to perform that operation.

```
# Import the required modules
from bankws import c2b
from bankws import c2bhelper
from bankws import entity
from bankws import webservice
from bankws import bank

# Create the company which originates the payment
company = entity.Entity('Firma Oy', 'FI', 'Teollisuuskatu 1', '00550', 'Helsinki')

# Create the payment receiver
receiver = entity.Entity('Ville Velkoja', 'FI', 'Rahakatu 71', '26458', 'Pankkila')

# create the C2B-payment using the payer as an argument
payment = c2b.C2B(company)

# Add group header to identify payment and MIXD to tell, that there is one or more of
# PmtInf-sections holding one or more of CdtTrfTxInf-sections.
payment.group_header('test_payment_4', 'MIXD')

# Create individual payment using the following information. See the documentation for
# the meaning of different parameters.
sepa = c2bhelper.SEPA(company, 'TRA', 'SEPA', '2012-05-28',
                   'material-id_7', 'FI2550001520322972', 'OKOYFIHH',
                   'SLEV', 'pmt_inf_id', 'instr_prty', 'EUR', 'Toinen Firma Oy')

# Add the “Maksun hyvityspuolen tiedot” to the payment. Check the start of this document
# for the meaning of different parameters.
tr = c2bhelper.CdtTrfTxInf(receiver, None, 'unique_transaction_7', 8000.01,
                          'EUR', 'OKOYFIHH', 'FI8937040044053000', 'SLEV',
                          'unique_transaction_7 (InstrId)', 'HIGH',
                           None, 'Testiviesti vastaanottajalle', None)

# Add the transaction to the list of transactions in a payment
sepa.add_transaction(tr1)

# Add payment to the list of individual payments in a message
payment.add_SEPA_payment(sepa)

# Create the webservice-object with a customer’s ID, key-and bank’s information.
ws = webservice.WebService(CustomerID, RSA keyfile name, Certificate file name, Bank object)

# Upload file to the bank
pmt_xml = str(payment)
ar = ws.upload_file(pmt_xml)

# Print the payment’s reference number returned by the bank later used when downloading
# the filelist. This output number is used as a parent reference-number when selecting
# the appropriate feedback to be downloaded regarding this particular message
print(ar.references[0])
```