python-en16931 Documentation

Release 0.0.1

Invinet Sistemes

Jul 11, 2018

Contents

Python 3 package to read, write and manage the new EN16931 Invoice format.

This European Standard establishes a semantic data model of the core elements of an electronic invoice. The semantic model includes only the essential information elements that an electronic invoice needs to ensure legal (including fiscal) compliance and to enable interoperability for cross-border, cross sector and for domestic trade.

1 Features

This library allows you to:

1. De-serialize an XML in EN16931 format to a python Invoice object.
2. Serialize a python Invoice object to a valid XML representation.
3. Validate an Invoice using validex.
4. Import an Invoice to B2BRouter.

2 Usage

You can import an invoice from an XML file:

```
>>> from en16931 import Invoice
>>> invoice = Invoice.from_xml('en16931/tests/files/invoice.xml')
```

And use the API to access its internal values and entities:

```
>>> invoice.issue_date
datetime.datetime(2018, 6, 11, 0, 0)
>>> invoice.seller_party
<en16931.entity.Entity at 0x7f2b7c12b860>
>>> invoice.buyer_party
```

(continues on next page)

(continued from previous page)

```
<en16931.entity.Entity at 0x7f2b7c0fd160>
>>> invoice.unique_taxes
{ Tax S: 0.21  , Tax S: 0.1  }
>>> invoice.lines
[<en16931.invoice_line.InvoiceLine at 0x7f2b7c0fd400>,
 <en16931.invoice_line.InvoiceLine at 0x7f2b7c0fd518>,
 <en16931.invoice_line.InvoiceLine at 0x7f2b7c0fd748>]
>>> invoice.tax_exclusive_amount
87.00
>>> invoice.tax_amount()
16.62
>>> invoice.tax_inclusive_amount
103.62
>>> invoice.payable_amount
103.62
```

If you import an XML file, all relevant quantities are not computed; we use the ones defined on the XML. You can check that the computed and imported quantities match by calling the relevant methods:

```
>>> assert invoice.tax_exclusive_amount == invoice.subtotal()
True
>>> assert invoice.tax_inclusive_amount == invoice.total()
True
>>> assert invoice.payable_amount == invoice.total()
True
```

Or you can also build, step by step, an invoice:

```
>>> from en16931 import Invoice
>>> invoice = Invoice(invoice_id="2018-01", currency="EUR")
>>> seller = Entity(name="Acme Inc.", tax_scheme="VAT",
...                 tax_scheme_id="ES34626691F", country="ES",
...                 party_legal_entity_id="ES34626691F",
...                 registration_name="Acme INc.", mail="acme@acme.io",
...                 endpoint="ES76281415Y", endpoint_scheme="ES:VAT",
...                 address="easy street", postalzone="08080",
...                 city="Barcelona")
>>> buyer = Entity(name="Corp Inc.", tax_scheme="VAT",
...                tax_scheme_id="ES76281415Y", country="ES",
...                party_legal_entity_id="ES76281415Y",
...                registration_name="Corp INc.", mail="corp@corp.io",
...                endpoint="ES76281415Y", endpoint_scheme="ES:VAT",
...                address="busy street", postalzone="08080",
...                city="Barcelona")
>>> invoice.buyer_party = buyer
>>> invoice.seller_party = seller
>>> invoice.due_date = "2018-09-11"
>>> invoice.issue_date = "2018-06-11"
>>> # lines
>>> il1 = InvoiceLine(quantity=11, unit_code="EA", price=2,
...                   item_name='test 1', currency="EUR",
...                   tax_percent=0.21, tax_category="S")
>>> il2 = InvoiceLine(quantity=2, unit_code="EA", price=25,
...                   item_name='test 2', currency="EUR",
...                   tax_percent=0.21, tax_category="S")
>>> il3 = InvoiceLine(quantity=5, unit_code="EA", price=3,
```

(continues on next page)

(continued from previous page)

```
...                     item_name='test 3', currency="EUR",
...                     tax_percent=0.1, tax_category="S")
>>> invoice.add_lines_from([il1, il2, il3])
```

And serialize it to XML:

```
>>> # As a string
>>> xml = invoice.to_xml()
>>> # Or save it directly to a file
>>> invoice.save('example_invoice.xml')
```

3 Limitations

This is a proof of concept implementation and not all features defined in the EN16931 standard are implemented. But it is easy, in some cases trivial, to implement them. The main not implemented features are:

- CreditNotes are not supported.
- File attachments are not supported.
- Delivery information is not supported.
- Only global charges and discounts are supported. Line discounts and charges are not supported.
- Other potentially useful attributes (such as InvoicePeriod, BuyerReference, OrderReference, BillingReference, ContractDocumentReference, among others) are not implemented.

If you need a particular feature implemented, see the following section for feature requests.

4 Bugs and Feature Requests

Please report any bugs that you find here. Or, even better, fork the repository on GitHub and create a pull request (PR). We welcome all changes, big or small.

5 License

Released under the Apache License Version 2.0 (see *LICENSE.txt*):

```
Copyright (C) 2018 Invinet Sistemes
```

6 Reference

6.1 Classes

Invoice

class en16931.**Invoice**(*invoice_id=None*, *currency='EUR'*, *from_xml=False*)

EN16931 Invoice class.

This is the main entry point of this library. You can build, step by step, an Invoice and then serialize to XML.

It uses the class *Entity* to represent seller and buyer parties, and the class *InvoiceLine* to represent invoice lines.

```
>>> from en16931 import Invoice
>>> invoice = Invoice(invoice_id="2018-01", currency="EUR")
>>> seller = Entity(name="Acme Inc.", tax_scheme="VAT",
...                 tax_scheme_id="ES34626691F", country="ES",
...                 party_legal_entity_id="ES34626691F",
...                 registration_name="Acme INc.", mail="acme@acme.io",
...                 endpoint="ES76281415Y", endpoint_scheme="ES:VAT",
...                 address="easy street", postalzone="08080",
...                 city="Barcelona")
>>> buyer = Entity(name="Corp Inc.", tax_scheme="VAT",
...                tax_scheme_id="ES76281415Y", country="ES",
...                party_legal_entity_id="ES76281415Y",
...                registration_name="Corp INc.", mail="corp@corp.io",
...                endpoint="ES76281415Y", endpoint_scheme="ES:VAT",
...                address="busy street", postalzone="08080",
...                city="Barcelona")
>>> invoice.buyer_party = buyer
>>> invoice.seller_party = seller
>>> invoice.due_date = "2018-09-11"
>>> invoice.issue_date = "2018-06-11"
>>> # lines
>>> il1 = InvoiceLine(quantity=11, unit_code="EA", price=2,
...                   item_name='test 1', currency="EUR",
...                   tax_percent=0.21, tax_category="S")
>>> il2 = InvoiceLine(quantity=2, unit_code="EA", price=25,
...                   item_name='test 2', currency="EUR",
...                   tax_percent=0.21, tax_category="S")
>>> il3 = InvoiceLine(quantity=5, unit_code="EA", price=3,
...                   item_name='test 3', currency="EUR",
...                   tax_percent=0.1, tax_category="S")
>>> invoice.add_lines_from([il1, il2, il3])
```

And serialize it to XML:

```
>>> # As a string
>>> xml = invoice.to_xml()
>>> # Or save it directly to a file
>>> invoice.save('example_invoice.xml')
```

__init__(*invoice_id=None*, *currency='EUR'*, *from_xml=False*)

Initialize an Invoice.

This is the main class and entry point for creating an Invoice.

Parameters

- **invoice_id** (*string (optional, default '1')*) – Arbitrary string to identify the invoice.
- **currency** (*string (optional, default 'EUR')*) – An ISO 4217 currency code.
- **from_xml** (*bool (optional, default False)*) – A flag to mark if the object is the result of importing an XML invoice.

Raises KeyError: If the currency code is not a valid ISO 4217 code.

Examples

By default the currency of the invoice is EUR and its id is 1:

```
>>> i = Invoice()
>>> i.invoice_id
1
>>> i.currency
EUR
```

You can also specify an arbitrary id and a valid ISO 4217 currency code.

```
>>> i = Invoice(invoice_id="0001-2018", currency="USD")
>>> i.invoice_id
0001-2018
>>> i.currency
USD
```

__weakref__
list of weak references to the object (if defined)

add_line(*line*)
Adds an InvoiceLine to the Invoice.

Parameters **line** (*InvoiceLine object.*) –

add_lines_from(*container*)
Adds InvoiceLine instances from a container.

Parameters **container** (*container*) – An iterable container of InvoiceLine objects.

buyer_party
Property – The Entity with the role of AccountingCustomerParty.

See the *Entity* class for details

Parameters **party** (*Entity object.*) – The Entity object that plays the role of AccountingCustomerParty.

Raises

- ValueError – if the Entity is not valid.
- TypeError – if the input is not an Entity or Entity subclass.

charge
Property – The ChargeTotalAmount of the Invoice.

Parameters **value** (*string, integer, float*) – The input must be a valid input for the Decimal class the Python Standard Library.

Raises *decimal.InvalidOperation: If the input cannot be converted* – to a Decimal.

charge_base_amount
The base amount of the charge.

The BaseAmount of the charge in PEPPOL BIS 3 terms.

charge_percent
The percentage that the charge represents.

The MultiplierFactorNumeric of the charge in PEPPOL BIS 3 terms.

currency
Property – String representation of the ISO 4217 currency code.

Parameters **currency_str** (*string*) – String representation of the ISO 4217 currency code.

Raises KeyError: If the currency code is not a valid ISO 4217 code.

discount
Property – The AllowanceTotalAmount of the Invoice.

Parameters **value** (*string, integer, float*) – The input must be a valid input for the Decimal class the Python Standard Library.

Raises *decimal.InvalidOperation: If the input cannot be converted* – to a Decimal.

discount_base_amount
The base amount of the discount.

The BaseAmount of the discount in PEPPOL BIS 3 terms.

discount_percent
The percentage that the discount represents.

The MultiplierFactorNumeric of the discount in PEPPOL BIS 3 terms.

due_date
Property – Due date of the invoice.

Parameters **date** (*datetime or string*) – If the input is a string, it should be in one of the following formats: "%Y-%m-%d", "%Y%m%d", "%d-%m-%Y", "%Y/%m/%d", "%d/%m/%Y".

Raises `ValueError` – if the input string cannot be converted to a datetime object.

Examples

```
>>> from datetime import datetime
>>> i = Invoice()
```

Supported date formats are:

```
>>> i.due_date = datetime(2018, 6, 21)
>>> i.due_date
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.due_date = "2018-06-21"
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.due_date = "20180621"
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.due_date = "21-6-2018"
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.due_date = "2018/06/21"
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.due_date = "21/6/2018"
datetime.datetime(2018, 6, 21, 0, 0)
```

Incorrect date formats will raise a ValueError:

```
>>> i.due_date = "today"
Traceback (most recent call last):
[...]
ValueError: See documentation for string date formats supported
```

classmethod from_xml(*xml_path*)

Import a XML invoice in EN16931 format.

Parameters **xml_path** (*path*) – A path to the XML file.

Raises FileNotFoundError: if the file does not exist.

Examples

```
>>> i = Invoice.from_xml('path/to/invoice.xml')
```

gross_subtotal(*tax_type=None*)

Sum of gross amount of each invoice line.

issue_date

Property – The issue date of the invoice.

Parameters **date** (*datetime or string*) – If the input is a string, it should be in one of the following formats: "%Y-%m-%d", "%Y%m%d", "%d-%m-%Y", "%Y/%m/%d", "%d/%m/%Y".

Raises ValueError: – if the input string cannot be converted to a datetime object.

Examples

```
>>> from datetime import datetime
>>> i = Invoice()
```

Supported date formats are:

```
>>> i.issue_date = datetime(2018, 6, 21)
>>> i.issue_date
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.issue_date = "2018-06-21"
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.issue_date = "20180621"
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.issue_date = "21-6-2018"
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.issue_date = "2018/06/21"
datetime.datetime(2018, 6, 21, 0, 0)
>>> i.issue_date = "21/6/2018"
datetime.datetime(2018, 6, 21, 0, 0)
```

Incorrect date formats will raise a ValueError:

```
>>> i.issue_date = "today"
Traceback (most recent call last):
[...]
ValueError: See documentation for string date formats supported
```

line_extension_amount
The total LineExtensionAmount of the invoice.

It's only computed as the *gross_subtotal()* if the Invoice was not imported from an XML file. In that case, its value is the one reported on the XML.

lines_with_taxes(*tax_type=None*)
Generator of InvoiceLines

Parameters **tax_type** (*Tax object (default None)*) – If a Tax object is provided, only generate lines with that Tax. If this parameter is None, generate all lines.

payable_amount
The total PayableAmount of the invoice.

It's only computed as the *total()* if the Invoice was not imported from an XML file. In that case, its value is the one reported on the XML.

payment_means_code
Property – The payment means code.

It has to be one of:

- '10': 'cash'
- '49': 'debit'
- '31': 'transfer'
- '26': 'cheque'
- '23': 'cheque_b'
- '48': 'credit'
- 'ZZZ': 'awarding, reposition, special'

Parameters **code** (*string*) – A valid payment means code.

Raises ValueError – If the code is not valid.

save(*path=None*)
Save the XML representation of the invoice.

Parameters **path** (*a path (optional, default None)*) – If the path is None it a file named 'invoice_id.xml' will be created in the current working directory.

seller_party
Property – The Entity with the role of AccountingSupplierParty.

See the *Entity* class for details

Parameters **party** (*Entity object.*) – The Entity object that plays the role of AccountingSupplierParty.

Raises

- ValueError: – if the Entity is not valid.
- TypeError: – if the input is not an Entity or Entity subclass.

subtotal(*tax_type=None*)
Gross amount before taxes.

TotalGrossAmount - AllowanceTotalAmount + ChargeTotalAmount

tax_amount(*tax_type=None*)
Computes the tax amount of the Invoice.

Parameters **tax_type** (*Tax object (default None)*) – If a Tax object is provided, the tax amount corresponding to the porvided Tax. If None the total tax amount.

tax_exclusive_amount
The total TaxExclusiveAmount of the invoice.

It's only computed as the *gross_subtotal()* if the Invoice was not imported from an XML file. In that case, its value is the one reported on the XML.

tax_inclusive_amount
The total TaxInclusiveAmount of the invoice.

It's only computed as the *total()* if the Invoice was not imported from an XML file. In that case, its value is the one reported on the XML.

taxable_base(*tax_type=None*)
Computes the taxable base of the Invoice

Parameters **tax_type** (*Tax object (default None)*) – If a Tax object is provided, the taxable base corresponding to the porvided Tax. If None the total taxable base.

to_xml()
Serialize the invoice object to XML.

Generate a valid PEPPOL BIS 3 XML document using the UBL 2.1 syntax.

total()
Computes the TaxInclusiveAmount of the Invoice

unique_taxes
Set of unique taxes in the Invoice.

Invoice Line

class en16931.**InvoiceLine**(*quantity=None*, *unit_code='EA'*, *price=None*, *item_name=None*, *currency='EUR'*, *tax_percent=None*, *line_extension_amount=None*, *tax_category=None*, *tax_name=None*)
EN16931 InvoiceLine class.

Each *Invoice* has to have at least one invoice line in which the quantity and the price of the items is reflected.

You can initialize an InvoiceLine instance with all its attributes:

```
>>> il = InvoiceLine(quantity=11, unit_code="EA", price=2,
...                  item_name='test', currency="EUR",
...                  tax_percent=0.21, tax_category="S")
```

Or cou can do it step by step:

```
>>> il = InvoiceLine()
>>> il.quantity = 11
>>> il.price = 2
>>> il.item_name = 'test'
>>> il.tax_percent = 0.21
>>> il.tax_category = "S"
```

An InvoiceLine is only valid if it has quantity, price and tax defined:

```
>>> il.is_valid()
True
>>> new_line = InvoiceLine()
>>> new_line.is_valid()
False
```

__init__(*quantity=None*, *unit_code='EA'*, *price=None*, *item_name=None*, *currency='EUR'*, *tax_percent=None*, *line_extension_amount=None*, *tax_category=None*, *tax_name=None*)

Initialize an Invoice Line.

Parameters

- **quantity** (*float or integer.*) – The number of items of the line.
- **unit_code** (*string (optional)*) – A unit code defining the nature of the quantities of the items of the line. It must be one of: 'EA': 'units', 'HUR': 'hours', 'KGM': 'kilograms', 'LTR': 'litters', 'DAY': 'days', 'CS': 'boxes'.
- **price** (*string, integer, float*) – The input must be a valid input for the Decimal class the Python Standard Library.
- **item_name** (*string (optional)*) – Arbitrary name to define the item of the line.
- **currency** (*string.*) – String representation of the ISO 4217 currency code.
- **tax_percent** (*float.*) – The percentage of the Tax applied to the line. Can be 0.
- **tax_category** (*string.*) – A string representing the category of the Tax. It must be one of 'AE', 'L', 'M', 'E', 'S', 'Z', 'G', 'O', or 'K'.
- **tax_name** (*string.*) – Arbitrary name to identify the Tax.
- **line_extension_amount** (*string, integer, float*) – The input must be a valid input for the Decimal class the Python Standard Library. Computed unless the invoice is imported from an XML file.

Notes

An InvoiceLine is considered valid if and only if it has quantity, price and tax.

__weakref__

list of weak references to the object (if defined)

currency

Property – String representation of the ISO 4217 currency code.

Parameters **currency_str** (*string*) – String representation of the ISO 4217 currency code.

Raises KeyError: If the currency code is not a valid ISO 4217 code.

has_tax(*tax*)

Returns True if the line has this tax.

Parameters **tax** (*Tax Object.*) –

is_valid()

Returns True if the line is valid.

item_name

The arbitrary name of the item of the line.

Parameters **item_name** (*string (optional)*) – Arbitrary name to define the item of the line.

line_extension_amount
Property – The LineExtensionAmount

Parameters **line_extension_amount** (*string, integer, float*) – The input must be a valid input for the Decimal class the Python Standard Library. Computed unless the invoice is imported from an XML file.

price
Property – The price of one item.

Parameters **price** (*string, integer, float*) – The input must be a valid input for the Decimal class the Python Standard Library.

quantity
Property – Quantity of items of the line.

Parameters **quantity** (*float or integer.*) – The number of items of the line.

tax
Returns a Tax object representing the taxes applied to the line.

unit_code
property – The unit code defining the nature of the quantities.

Parameters **unit_code** (*string.*) – A unit code defining the nature of the quantities of the items of the line. It must be one of: 'EA': 'units', 'HUR': 'hours', 'KGM': 'kilograms', 'LTR': 'litters', 'DAY': 'days', 'CS': 'boxes'.

Entity

class en16931.**Entity**(*name=None*, *tax_scheme=None*, *tax_scheme_id=None*, *country=None*, *party_legal_entity_id=None*, *registration_name=None*, *mail=None*, *endpoint=None*, *endpoint_scheme=None*, *postalzone=None*, *city=None*, *address=None*)

Entity class.

It represents a party involved in the trade described by the invoice, such as seller or buyer party.

You can initialize an Entity with most of its attributes:

```
>>> e = Entity(name="Acme Inc.", tax_scheme="VAT",
...            tax_scheme_id="ES34626691F", country="ES",
...            party_legal_entity_id="ES34626691F",
...            registration_name="Acme INc.", mail="acme@acme.io",
...            endpoint="ES76281415Y", endpoint_scheme="ES:VAT",
...            address="easy street", postalzone="08080",
...            city="Barcelona")
```

Or you can build it step by step:

```
>>> e.name = "Acme Inc."
>>> e.tax_scheme = "VAT"
>>> e.tax_scheme_id = "ES34626691F"
>>> e.country = "ES"
>>> e.party_legal_entity_id = "ES34626691F"
>>> e.registration_name = "Acme INc."
>>> e.endpoint = "ES76281415Y"
```

(continues on next page)

(continued from previous page)

```
>>> e.endpoint_scheme = "ES:VAT"
>>> p = PostalAddress(address="easy street", city_name="Barcelona",
...                   postal_zone="08080", country="ES")
>>> e.postal_address = p
```

You can assign a *PostalAddress* to its *postal_address()* property. You can also associate a *BankInfo* instance to an Entity in order to store relevant banking information for debit and transfer payments:

```
>>> bank_info = BankInfo(iban="ES661234563156", bic="AAAABBCCDDD")
>>> e.bank_info = bank_info
>>> e.bank_info.iban
"ES661234563156"
```

An entity is valid if it has a name, a country, valid ids, valid taxscheme and endpoint, and has an address.

```
>>> e.is_Valid()
True
```

__init__(*name=None*, *tax_scheme=None*, *tax_scheme_id=None*, *country=None*, *party_legal_entity_id=None*, *registration_name=None*, *mail=None*, *endpoint=None*, *endpoint_scheme=None*, *postalzone=None*, *city=None*, *address=None*)

Initialize an Entity.

TODO formal definition of Entity.

Parameters

- **name** (*string.*) – The name of the Entity.
- **tax_scheme** (*string.*) – The tax scheme of the Entity.
- **tax_scheme_id** (*string.*) – The tax ID of the Entity.
- **country** (*string.*) – Two letter code for the country of the Entity.
- **party_legal_entity_id** (*string.*) – The party legal entity of the Entity.
- **registration_name** (*string.*) – The Registration name of the Entity.
- **mail** (*string.*) – The contact Email of the Entity.
- **endpoint** (*string.*) – A valid PEPPOL endpoint.
- **endpoint_scheme** (*string.*) – The scheme defining the endpoint.
- **postalzone** (*string.*) – The postalzone of the address of the Entity.
- **city** (*string.*) – The city of the address of the Entity.
- **address** (*string.*) – The address of the Entity.

Notes

An entity is valid if it has a name, a country, valid ids, valid taxscheme and endpoint, and has an address.

__weakref__

list of weak references to the object (if defined)

bank_info

Property – a *BankInfo* instance

Parameters **bank_info** (*BankInfo instance*) – a valid BankInfo instance.

Raises

- `TypeError` – If the parameter is not a BankInfo instance
- `ValueError` – If the BankInfo instance is not valid

country
Property – The country of the entity.

Parameters **country** (*string.*) – Two letter code for the country of the Entity.

endpoint
Property – The endpoint ID of the Entity.

Parameters **endpoint** (*string.*) – A valid PEPPOL endpoint.

endpoint_scheme
Property – The endpoint scheme of the Entity.

Parameters **endpoint_scheme** (*string.*) – The scheme defining the endpoint.

is_valid()
Returns True if the Entity is valid.

An entity is valid if it has a name, a country, valid ids, valid taxscheme and endpoint, and has an address.

mail
The contact mail of the Entity.

name
Property – The name of the Entity.

Parameters **name** (*string.*) – The name for the Entity.

party_legal_entity_id
The party legal entity ID

postal_address
Property – The PostalAddress of the Entity.

See the *PostalAddress* class.

Parameters **address** (*PostalAddress object.*) – A PostalAddress instance.

Raises *TypeError: if the input is not a PostalAddreess* – or a subclass.

registration_name
The registration name of the Entity.

tax_scheme
Property – The tax scheme of the Entity.

Parameters **scheme** (*string.*) – The tax scheme used by the Entity.

Raises ValueError: if the tax scheme is not valid.

tax_scheme_id
Property – The tax ID of the Entity.

Parameters **tax_scheme_id** (*string.*) – The tax ID of the Entity.

BankInfo

class en16931.**BankInfo**(*account=None*, *iban=None*, *bic=None*, *mandate_reference_identifier=None*)

BankInfo class.

Stores relevant banking information of *Entity* to specify the needed information when the payment means of the invoice involves a Bank.

You can initialize a BankInfo instance with all needed attributes:

```
>>> b = BankInfo(account="1234567321", bic="AAAABBCCDDD",
...              mandate_reference_identifier="123")
```

Or build it, step by step:

```
>>> b = BankInfo()
>>> b.account = "1234567321"
>>> b.bic = "AAAABBCCDDD"
>>> b.mandate_reference_identifier = "123"
```

For the bank information to be complete it has to contain the IBAN number, or the bank account and the BIC. The mandate reference identifier is used in the context of debit payments.

```
>>> b.is_valid()
True
```

__init__(*account=None*, *iban=None*, *bic=None*, *mandate_reference_identifier=None*)

Initialize the Bank Information for an Entity.

Parameters

- **account** (*string (optional)*) – The bank account number.
- **bic** (*string (optional)*) – The Bank Identification Code of the account.
- **iban** (*string (optional)*) – The International Bank Account Number (IBAN)
- **mandate_reference_identifier** (*string (optional)*) – The Mandate Reference Identifier

Notes

For the bank information to be complete it has to contain the IBAN number, or the bank account and the BIC. The mandate reference identifier is used in the context of debit payments.

__weakref__

list of weak references to the object (if defined)

account

Property – Bank account number

Parameters **account** (*string*) – The bank account number.

Notes

No validation of the bank account number is performed.

bic
Property – The Bank Identification Code

Also known as SWIFT code.

Parameters **bic** (*string*) – The Bank Identification Code of the account.

Raises ValueError – If the BIC code is not valid.

Notes

A BIC code has either 8 or 11 characters without dashes and spaces.

has_mandate_reference()
Returns True if it has a mandate reference identifier.

This is necessary for debit payments.

iban
Property – The International Bank Account Number (IBAN)

Parameters **iban** (*string*) – The International Bank Account Number (IBAN)

Notes

No validation of the IBAN is performed.

is_valid()
Returns True if the information is complete.

A valid bank information for an *Entity* must have a bank account and a BIC or an IBAN.

mandate_reference_identifier
Property – The Mandate Reference Identifier

The Mandate Reference Identifier is necessary for debit payments.

Parameters **mandate_reference_identifier** (*string*) – The Mandate Reference Identifier

Postal Address

class en16931.**PostalAddress**(*address=None*, *city_name=None*, *postal_zone=None*, *country=None*)
PostalAddress class

It represents a postal address of an *Entity*.

__init__(*address=None*, *city_name=None*, *postal_zone=None*, *country=None*)
Initializes a PostalAddress.

Parameters

- **address** (*string.*) – An address.
- **city_name** (*string.*) – The name of a city.
- **postal_zone** (*string.*) – A valid postal zone.
- **country** (*string.*) – A valid two letter country code.

__weakref__
list of weak references to the object (if defined)

country
The country of the address

Tax

class en16931.**Tax**(*percent*, *category*, *name*, *comment=''*)
Tax class.

It representas a tax to apply globally or to a concrete invoice line.

Only categories of taxes enabled by the EN16931 standard are supported. See the documentation of *category()* property for more details.

You can create Tax objects directly:

```
>>> t = Tax(0.21, "S", "IVA")
```

Or specify the relevant attributes when building InvoiceLines or *Invoice*

__eq__(*other*)
A tax is compared to other Tax objects by equality of their percentage, category, and name.

__hash__()
Return hash(self).

__init__(*percent*, *category*, *name*, *comment=''*)
Initialize a Tax object.

Parameters

- **category** (*string.*) – A string representing the category of the Tax. It must be one of 'AE', 'L', 'M', 'E', 'S', 'Z', 'G', 'O', or 'K'.
- **percent** (*float.*) – The percentage of the Tax. Can be 0.
- **name** (*string.*) – Arbitrary name to identify the Tax.
- **comment** (*string.*) – A comment on the tax.

Notes

A tax is compared to other Tax objects by equality of their percentage, category, and name.

__repr__()
Return repr(self).

__weakref__
list of weak references to the object (if defined)

category
Property – The category of the Tax.

Parameters **category** (*string.*) – A string representing the category of the Tax. It must be one of 'AE', 'L', 'M', 'E', 'S', 'Z', 'G', 'O', or 'K'.

Raises ValueError: if the category is not valid.

code
An identification code of the tax.

6.2 Modules

b2brouter

Module to interact with b2brouter.net

en16931.b2brouter.**post_to_b2brouter**(*invoice*, *api_key*, *project_id*, *test=False*)
Posts an Invoice to b2brouter.net

Parameters

- **api_key** (*string.*) – The authentification API key for b2brouter.net
- **project_id** (*string.*) – The project ID to which submit the invoice in b2brouter.net

xpaths

Manipulate and parse XML files

en16931.xpaths.**en16931_namespaces**()
Namespaces for the en16931 invoice format

en16931.xpaths.**en16931_xpaths**()
Xpaths for the en16931 invoice format.

en16931.xpaths.**get_charge**(*root*, *namespaces=None*, *xpaths=None*)
Gets the charge of an Invoice

en16931.xpaths.**get_discount**(*root*, *namespaces=None*, *xpaths=None*)
Gets the discount of an Invoice

en16931.xpaths.**get_entity**(*root*, *kind='seller'*)
Gets an Entity of an Invoice

en16931.xpaths.**get_from_xpath**(*root*, *tag*, *xpaths=None*, *namespaces=None*)
Gets the content of an XPATH in an XML file.

en16931.xpaths.**get_invoice_lines**(*root*, *namespaces=None*)
Generator of InvoiceLines of an Invoice

en16931.xpaths.**get_namespaces**()
Get a dictionari with all namespaces.

en16931.xpaths.**get_xpaths**()
Get a dictionary with all xpaths

utils

Miscelanious util functions

validex

Module to interact with open.validex.net

You need to create an user at validex.net to be able to use its API.

en16931.validex.**is_valid_at_validex**(*invoice*, *api_key*, *user_id*)
Validates an Invoice at open.validex.net

You need to create an user at validex.net to be able to use its API.

Parameters

- **api_key** (*string.*) – The authentification API key for validex.net
- **user_id** (*string.*) – The user ID of validex.net

Notes

Warnings are not reported.

7 Indices and tables

- genindex
- modindex
- search

Python Module Index

e