Commercetools Python SDK Documentation

Release 13.0.0

Lab Digital

Jan 20, 2021

Usage Documentation

This is an unofficial Python SDK for the Commercetools platform. It only supports Python 3.6+ and uses type annotation for an improved development experience.

The API is mostly generated using the commercetools api RAML file and uses the attr library for the dataobjects and marshmallow for the serialization and deserialization steps.

CHAPTER 1

Installation

```
pip install commercetools
```

CHAPTER 2

Example

```
from commercetools import Client

client = Client(
    project_key="<your-project-key>",
    client_id="<your-client-id>",
    client_secret="<your-client-secret>",
    scope=["<scopes>"],
    url="https://api.sphere.io",
    token_url="https://auth.sphere.io/oauth/token",
)

product = client.products.get_by_id("00633d11-c5bb-434e-b132-73f7e130b4e3")
print(product)
```

The client can also be configured by setting the following environment variables:

```
export CTP_PROJECT_KEY="<project key>"
export CTP_CLIENT_SECRET="<client secret>"
export CTP_CLIENT_ID="<client id>"
export CTP_AUTH_URL="https://auth.sphere.io"
export CTP_API_URL="https://api.sphere.io"
export CTP_SCOPES="<comma seperated list of scopes>"
```

And then constructing a client without arguments:

```
from commercetools import Client

client = Client()

product = client.products.get_by_id("00633d11-c5bb-434e-b132-73f7e130b4e3")
print(product)
```

2.1 The Client class

class commercetools.**Client**(**args*, ***kwargs*)
Bases: commercetools.base_client.BaseClient, commercetools.services.ServicesMixin

Note: The classes listed below should generally not be instantiated directly but instead be accessed via the *commercetools.Client()* class.

2.1.1 Client.carts

class commercetools.services.carts.**CartService**(*client*)
Bases: commercetools.services.abstract.AbstractService

A shopping cart holds product variants and can be ordered.

create(*draft*, *, *expand=None*)
Creating a cart can fail with an InvalidOperation if the referenced shipping method in the

CartDraft has a predicate which does not match the cart. A shopping cart holds product variants and can be ordered.

Return type Cart

delete_by_id(*id*, *version*, *, *expand=None*, *data_erasure=None*, *force_delete=False*)

Return type Cart

get_by_customer_id(*customer_id*, *, *expand=None*)
Retrieves the active cart of the customer that has been modified most recently.

It does not consider carts with CartOrigin Merchant. If no active cart exists, a 404 Not Found error is returned. The cart may not contain up- to-date prices, discounts etc. If you want to ensure they're up-to-date, send an Update request with the Recalculate update action instead.

Return type Cart

get_by_id(*id*, *, *expand=None*)
The cart may not contain up-to-date prices, discounts etc.

If you want to ensure they're up-to-date, send an Update request with the Recalculate update action instead.

Return type Cart

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*, *customer_id=None*)
A shopping cart holds product variants and can be ordered.

Return type CartPagedQueryResponse

replicate(*draft*)

Return type Cart

update_by_id(*id*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type Cart

2.1.2 Client.categories

class commercetools.services.categories.**CategoryService**(*client*)
Bases: commercetools.services.abstract.AbstractService

Categories are used to organize products in a hierarchical structure.

create(*draft*, *, *expand=None*)
Creating a category produces the CategoryCreated message.

Categories are used to organize products in a hierarchical structure.

Return type Category

delete_by_id(*id*, *version*, *, *expand=None*, *force_delete=False*)

Return type Category

delete_by_key(*key*, *version*, *, *expand=None*, *force_delete=False*)

Return type Category

get_by_id(*id*, *, *expand=None*)

Return type Category

get_by_key(*key*, *, *expand=None*)

Return type Category

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*)
Categories are used to organize products in a hierarchical structure.

Return type CategoryPagedQueryResponse

update_by_id(*id*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type Category

update_by_key(*key*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type Category

2.1.3 Client.channels

class commercetools.services.channels.**ChannelService**(*client*)
Bases: commercetools.services.abstract.AbstractService

Channels represent a source or destination of different entities.

They can be used to model warehouses or stores.

create(*draft*, *, *expand=None*)
Channels represent a source or destination of different entities. They can be used to model warehouses or stores.

Return type Channel

delete_by_id(*id*, *version*, *, *expand=None*, *force_delete=False*)

Return type Channel

get_by_id(*id*, *, *expand=None*)

Return type Channel

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*)
Channels represent a source or destination of different entities. They can be used to model warehouses or stores.

Return type ChannelPagedQueryResponse

update_by_id(*id*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type Channel

2.1.4 Client.custom_objects

class commercetools.services.custom_objects.**CustomObjectService**(*client*)
Bases: commercetools.services.abstract.AbstractService

Store custom JSON values.

create(*draft*, *, *expand=None*)
Creates a new custom object or updates an existing custom object.

If an object with the given container/key exists, the object will be replaced with the new value and the version is incremented. If the request contains a version and an object with the given container/key exists then the version must match the version of the existing object. Concurrent updates for the same custom object still can result in a Conflict (409) even if the version is not provided. Fields with null values will not be saved. Store custom JSON values.

Return type CustomObject

create_or_update(*draft*, *, *expand=None*)
Creates a new custom object or updates an existing custom object.

If an object with the given container/key exists, the object will be replaced with the new value and the version is incremented. If the request contains a version and an object with the given container/key exists then the version must match the version of the existing object. Concurrent updates for the same custom object still can result in a Conflict (409) even if the version is not provided. Fields with null values will not be saved. Store custom JSON values.

Return type CustomObject

delete_by_container_and_key(*container*, *key*, *, *data_erasure=None*, *version=None*, *expand=None*, *force_delete=False*)
Delete CustomObject by container and key

Return type CustomObject

get_by_container_and_key(*container*, *key*, *, *expand=None*)
Get CustomObject by container and key

Return type CustomObject

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*)
The query endpoint allows to retrieve custom objects in a specific container or all custom objects.

For performance reasons, it is highly advisable to query only for custom objects in a container by using the container field in the where predicate. Store custom JSON values.

Return type CustomObjectPagedQueryResponse

query_by_container(*container*, *, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*)
Store custom JSON values.

Return type CustomObjectPagedQueryResponse

2.1.5 Client.inventory

2.1.6 Client.orders

class commercetools.services.orders.**OrderService**(*client*)
Bases: commercetools.services.abstract.AbstractService

An order can be created from a order, usually after a checkout process has been completed.

create(*draft*, *, *expand=None*)
Creates an order from a Cart.

The cart must have a shipping address set before creating an order. When using the Platform TaxMode, the shipping address is used for tax calculation. An order can be created from a order, usually after a checkout process has been completed.

Return type Order

delete_by_id(*id*, *version*, *, *expand=None*, *data_erasure=None*, *force_delete=False*)

Return type Order

delete_by_order_number(*order_number*, *version*, *, *expand=None*, *data_erasure=None*, *force_delete=False*)

Return type Order

get_by_id(*id*, *, *expand=None*)

Return type Order

get_by_order_number(*order_number*, *, *expand=None*)
In case the orderNumber does not match the regular expression [a-zA-Z0-9_-]+,

it should be provided in URL-encoded format.

Return type Order

import_(*draft*)
Create an Order by Import

Return type Order

order_edit_apply(*id*, *action*)

Return type OrderEdit

order_edit_create(*draft*, *, *expand=None*)
OrderEdit are containers for financial changes after an Order has been placed.

Return type OrderEdit

order_edit_delete_by_id(*id*, *version*, *, *expand=None*, *force_delete=False*)

Return type OrderEdit

order_edit_delete_by_key(*key*, *version*, *, *expand=None*, *force_delete=False*)

Return type OrderEdit

order_edit_get_by_id(*id*, *, *expand=None*)

Return type OrderEdit

order_edit_get_by_key(*key*, *, *expand=None*)

Return type OrderEdit

order_edit_query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*)

OrderEdit are containers for financial changes after an Order has been placed.

Return type OrderEditPagedQueryResponse

order_edit_update_by_id(*id*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type OrderEdit

order_edit_update_by_key(*key*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type OrderEdit

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*)

An order can be created from a order, usually after a checkout process has been completed.

Return type OrderPagedQueryResponse

update_by_id(*id*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type Order

update_by_order_number(*order_number*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type Order

2.1.7 Client.payments

class commercetools.services.payments.**PaymentService**(*client*)

Bases: commercetools.services.abstract.AbstractService

Payments hold information about the current state of receiving and/or refunding money

create(*draft*, *, *expand=None*)

To create a payment object a payment draft object has to be given with the request.

Payments hold information about the current state of receiving and/or refunding money

Return type Payment

delete_by_id(*id*, *version*, *, *expand=None*, *data_erasure=None*, *force_delete=False*)

Return type Payment

delete_by_key(*key*, *version*, *, *expand=None*, *data_erasure=None*, *force_delete=False*)

Return type Payment

get_by_id(*id*, *, *expand=None*)

Return type Payment

get_by_key(*key*, *, *expand=None*)

Return type Payment

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*)

Payments hold information about the current state of receiving and/or refunding money

Return type PaymentPagedQueryResponse

update_by_id(*id*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type Payment

update_by_key(*key*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type Payment

2.1.8 Client.product_projections

class commercetools.services.product_projections.**ProductProjectionService**(*client*)
Bases: commercetools.services.abstract.AbstractService

A projected representation of a product shows the product with its current or staged data.

The current or staged representation of a product in a catalog is called a product projection.

get_by_id(*id*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*, *staged=None*)
Gets the current or staged representation of a product in a catalog by ID.

When used with an API client that has the view_published_products:{projectKey} scope, this endpoint only returns published (current) product projections.

Return type ProductProjection

get_by_key(*key*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*, *staged=None*)
Gets the current or staged representation of a product found by Key.

When used with an API client that has the view_published_products:{projectKey} scope, this endpoint only returns published (current) product projections.

Return type ProductProjection

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*, *staged=None*)
You can use the product projections query endpoint to get the current or staged representations of Products.

When used with an API client that has the view_published_products:{projectKey} scope, this endpoint only returns published (current) product projections. A projected representation of a product shows the product with its current or staged data. The current or staged representation of a product in a catalog is called a product projection.

Return type ProductProjectionPagedQueryResponse

search(*mark_matching_variants*, *, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*, *expand=None*, *fuzzy=None*, *fuzzy_level=None*, *staged=None*, *filter=None*, *filter_facets=None*, *filter_query=None*, *facet=None*, *text=None*)
Search Product Projection

This endpoint provides high performance search queries over ProductProjections. The query result contains the ProductProjections for which at least one ProductVariant matches the search query. This means that variants can be included in the result also for which the search query does not match. To determine

which ProductVariants match the search query, the returned ProductProjections include the additional field isMatchingVariant.

Return type `ProductProjectionPagedSearchResponse`

suggest(*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *fuzzy=None*, *staged=None*, *search_keywords=None*)
The source of data for suggestions is the searchKeyword field in a product

Return type `ProductProjection`

2.1.9 Client.product_types

class commercetools.services.product_types.**ProductTypeService**(*client*)
Bases: commercetools.services.abstract.AbstractService

Product Types are used to describe common characteristics, most importantly common custom attributes,

of many concrete products.

create(*draft*, *, *expand=None*)
Product Types are used to describe common characteristics, most importantly common custom attributes, of many concrete products.

Return type `ProductType`

delete_by_id(*id*, *version*, *, *expand=None*, *force_delete=False*)

Return type `ProductType`

delete_by_key(*key*, *version*, *, *expand=None*, *force_delete=False*)

Return type `ProductType`

get_by_id(*id*, *, *expand=None*)

Return type `ProductType`

get_by_key(*key*, *, *expand=None*)

Return type `ProductType`

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*)
Product Types are used to describe common characteristics, most importantly common custom attributes, of many concrete products.

Return type `ProductTypePagedQueryResponse`

update_by_id(*id*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type `ProductType`

update_by_key(*key*, *version*, *actions*, *, *expand=None*, *force_update=False*)

Return type `ProductType`

2.1.10 Client.products

class commercetools.services.products.**ProductService**(*client*)
Bases: commercetools.services.abstract.AbstractService

Products are the sellable goods in an e-commerce project on CTP.

This document explains some design concepts of products on CTP and describes the available HTTP APIs for working with them.

create(*draft*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*)

To create a new product, send a representation that is going to become the initial staged representation

of the new product in the master catalog. If price selection query parameters are provided, the selected prices will be added to the response. Products are the sellable goods in an e-commerce project on CTP. This document explains some design concepts of products on CTP and describes the available HTTP APIs for working with them.

Return type `Product`

delete_by_id(*id*, *version*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*, *force_delete=False*)

Return type `Product`

delete_by_key(*key*, *version*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*, *force_delete=False*)

Return type `Product`

file_upload(*id*, *fh*, *, *filename=None*, *variant=None*, *sku=None*, *staged=None*)

Uploads a binary image file to a given product variant.

The supported image formats are JPEG, PNG and GIF.

Return type `Product`

get_by_id(*id*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*)

Gets the full representation of a product by ID.

Return type `Product`

get_by_key(*key*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*)

Gets the full representation of a product by Key.

Return type `Product`

query(*, *expand=None*, *sort=None*, *limit=None*, *offset=None*, *with_total=None*, *where=None*, *predicate_var=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*)

You can use the query endpoint to get the full representations of products.

REMARK: We suggest to use the performance optimized search endpoint which has a bunch functionalities, the query API lacks like sorting on custom attributes, etc. Products are the sellable goods in an e-commerce project on CTP. This document explains some design concepts of products on CTP and describes the available HTTP APIs for working with them.

Return type `ProductPagedQueryResponse`

update_by_id(*id*, *version*, *actions*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*, *force_update=False*)

Return type `Product`

update_by_key(*key*, *version*, *actions*, *, *expand=None*, *price_currency=None*, *price_country=None*, *price_customer_group=None*, *price_channel=None*, *locale_projection=None*, *store_projection=None*, *force_update=False*)

Return type Product

upload_image(*id*, *fh*, *, *filename=None*, *variant=None*, *sku=None*, *staged=None*)

Uploads a binary image file to a given product variant.

The supported image formats are JPEG, PNG and GIF.

Return type Product

2.1.11 Client.project

class commercetools.services.project.**ProjectService**(*client*)

Bases: commercetools.services.abstract.AbstractService

get()

Return type Order

update(*version*, *actions*, *, *force_update=False*)

Return type Project

2.2 Data objects

Note: The data objects are automatically generated using the ols RAML Commercetools RAML API documents.

2.3 Serialization/Deserialization

Note: This Marshmallow schemas are automatically generated based on the Commercetools RAML API documents.

2.4 Utils

class commercetools.utils.**BaseTokenSaver**

Bases: object

add_token(*client_id*, *scopes*, *token*)

get_token(*client_id*, *scopes*)

class commercetools.utils.**DefaultTokenSaver**

Bases: *commercetools.utils.BaseTokenSaver*

add_token(*client_id*, *scopes*, *token*)

classmethod clear_cache()

get_token(*client_id*, *scopes*)

storage

commercetools.utils.**fix_token_url**(*token_url*)

Ensure the token url has the right format.

Often clients only pass the base url instead of the complete token url, which gets confusing for users.

Return type str

2.5 14.0.0 (unreleased)

We moved our code generation to the code generation tool from Commercetools, see https://github.com/commercetools/rmf-codegen

Reason for this is two-fold:

1. We can now generate the code for the *imporapi* and the *ml* api specifications next to the *platform* sdk.
2. It makes it easier for us to keep the code generation in sync with changes to the API specification.

The major difference is that it now also use the request builder pattern, matching the SDK's for the other languages (e.g. TypeScript).

The package is backwards compatible for now, although deprecation warnings are shown.

2.6 13.0.0 (2021-01-04)

- Regenerate code (fixed not being able to parse a lot of errors, so upgrading is recommended) - Cart:
  - CartDiscountValueGiftLineItemDraft channel variables type changed ChannelReference -> ChannelResourceIdentifier
  - CartDiscount value field type changed CartDiscountValue -> CartDiscountValueDraft
  - CustomLineItemDraft custom field type changed CustomFields -> CustomFieldsDraft
  - TaxedPriceDraft total_net and total_gross fields type changed TypedMoneyDraft -> Money
  - Common: - Add back fraction_digits to Money draft classes
  - Errors: - Many commercetools errors can now be parsed instead of throwing a ValueError - Fixed incorrect typing of extension response errors
  - Me: - Added address_key to various billing/shipping address update actions
  - Product: - Add limit to ProductProjectionPagedSearchResponse - Add ProductVariantAddedMessage
  - Store: - Change supply_channels from ChannelResourceIdentifier to ChannelReference
  - Subscription: - ResourceDeletedDeliverySchema added data_erasure
- Minimum required dependencies now require requests_mock>=1.8.0 (it already didn't work without it)
- Testing customer group added changeName and setKey actions
- Use Black 19 for formatting generated code

2.7 12.0.2 (2020-11-27)

- Testing backend: Fix custom object mock interface

2.8 12.0.1 (2020-11-18)

- Testing backend: support 'in' for single values, f.e. 'orderState in ("Open")'

2.9 12.0.0 (2020-10-15)

- Regenerate types (commercetools-api-reference 5ebb3153)
- Removed get_by_container and replaced by query_by_container (it's a query endpoint, not a get custom object endpoint)

2.10 11.0.0 (2020-09-18)

- Regenerated types (custom objects get by id has been removed)
- **Fixed query parameters not being sent:** for example: delete's dataErasure or product price query priceCurrency where not being sent

2.11 10.0.2 (2020-09-08)

- Testing predicates now support 'in' syntax

2.12 10.0.1 (2020-09-04)

- Fix product upload_image missing product id

2.13 10.0.0 (2020-09-01)

Note this release has some breaking changes regarding optional arguments in Commercetools types.

- Regenerate types
- Mypy fixes
- Make types stricter on when they are optional and when they are not

2.14 9.0.0 (2020-08-14)

Note this release has some breaking changes regarding imports and a lot of code is now autogenerated.

- Services are now generated by the RAML specification
- Optimize import times for faster startup by only loading necessary schemas
- Breaking: rename *commercetools.schemas.** -> *commercetools._schema.** since they are considered an implementation detail
- Fix incorrect schema in various nested fields
- Code generation now works with Python 3.8

2.15 8.3.0 (2020-07-21)

- Testing backend request mock parameters were case insensitive, causing expanding to fail in some cases

2.16 8.2.0 (2020-07-20)

- Regenerate types (mainly Store distribution channels functionality)
- Store: distribution channel functionality including testing actions added

2.17 8.1.5 (2020-07-15)

- Fixed API extensions endpoints
- Testing: add product change/add price actions

2.18 8.1.4 (2020-06-11)

- Testing: use languages when creating store from draft

2.19 8.1.3 (2020-06-10)

- Testing: add setLanguages to store testing backend

2.20 8.1.2 (2020-06-09)

- Testing: Add upload_image handler for product testing service
- Throw ValueError when discriminator cannot be found

2.21 8.1.1 (2020-05-20)

- Testing: Fix publish action on mock backend; already published products got overwritten
- Testing: Automatically reset token cache when using commercetools pytest fixtures

2.22 8.1.0 (2020-05-01)

- Testing: Added product publish and set prices actions on mock backend.

2.23 8.0.0 (2020-04-28)

Regenerated Commercetools types (needed missing Discount state enums). The 'LoggedResource' no longer exists, so a lot of types got changed, but effectively still have the same attributes. For safety we're doing a major release.

- Testing: add support for updating product type sets in mock server

2.24 7.0.0 (20202-04-16)

Note this release has some breaking changes regarding optional typing. If something is required by Commercetools it is now also required when creating an instance of a type.

- Re-generated schemas and types
- Removed Optional type from get_by_id calls
- Correctly use Optional typing according to RAML source file
- Improved error information in CommercetoolsError object
- Testing: Add setAttribute and addVariant actions to product testing backend
- Testing: Add addAttributeDefinition action to testing backend
- Testing: Add addPayment action to order testing backend
- Testing: Check for unique values in testing backend
- Testing: Fixed Attribute return object in ProductTypes testing backend

2.25 6.2.1 (2020-03-24)

- Fix marshallow breaking on missing **kwargs argument (#76)

2.26 6.2.0 (2020-03-08)

- Fix Query Predicate chaining (#75)
- Update dependencies so that code generation works with Python 3.8
- Update GitHub Actions integration

2.27 6.1.0 (2020-01-13)

- Added nested query predicate support (#72)

2.28 6.0.0 (2020-01-13)

- Allow passing in the base auth URL when creating a Client
- Add multiple DELETE endpoints for various backends
- Implement mocked actions for Orders mock backend
- Fix mock backend not properly support '<>' operator
- Add in operator in queries (#73)
- Fix cart replicate function (#74)

2.29 5.0.0 (2019-11-05)

This is breaking change since the commercetools api specification is moving from the *Money* type to the *TypedMoney* type. You should generally be able to replace references to *Money* with *CentPrecisionMoney*.

- Sync with latest API definitions (dd371e12506af969b8edfeb1540eec1e8cd5ab9d)
- Updated to work with marshmallow > 3.0.0
- Implement set custom field for tests (note custom type checks are not implemented)

2.30 4.1.0 (2019-08-14)

- Fix Inventory Entry types *This may break Inventory related types*
- Add external_oauth field on project
- Small runserver fixes
- Mock ShippingMethod.setPredicate
- Mock State.setTransitions

2.31 4.0.0 (2019-07-04)

- *Breaking* Generate with new API definitions, main difference is Reference -> ResourceIdentifier type.

2.32 3.9.0 (2019-06-27)

- Implement testing/introspect token endpoint, thanks to @mbarga!

2.33 3.8.1 (2019-06-26)

- Small fixes if data was not initialized for mock shipping methods actions

2.34 3.8.0 (2019-06-26)

- Implement AddZone and RemoveZone actions for shipping methods

2.35 3.7.0 (2019-06-18)

- Added discount codes
- Added cart discounts

2.36 3.6.1 (2019-06-18)

- Fix small bug in reference expansion if value is None

2.37 3.6.0 (2019-06-17)

- Generate with new API definitions, biggest changes are Resource -> BaseResource and ResourceSchema -> BaseResourceSchema
- Add resource expansion to all applicable objects, see https://docs.commercetools.com/http-api#reference-expansion for details.

2.38 3.5.1 (2019-05-27)

- Remove WIP code on carts get_by_id call

2.39 3.5.0 (2019-05-23)

- Generate with new API definitions
- Add missing 'store' field to Order

2.40 3.4.0 (2019-05-21)

- Fix rating number type not being an integer for Review related types

2.41 3.3.0 (2019-05-20)

- Add State service

2.42 3.2.0 (2019-05-08)

- Add Store service

2.43 3.1.1 (2019-04-26)

- Fix test server not working inside docker by binding to 0.0.0.0 instead of localhost

2.44 3.1.0 (2019-04-26)

- Implement testing shipping method actions

2.45 3.0.0 (2019-04-25)

- Add Api clients service
- Testing server now autoreloads on python changes (using Werkzeug)
- Regenerate types from commercetools-api-reference (fixes typos, some new fields, adds extension response errors)
- Add X-Correlation-ID to exception class
- Mock X-Correlation-ID header in testing framework

2.46 2.5.2 (2019-04-17)

- Implement testing predicates gte and lte
- Implement inventory testing actions

2.47 2.5.1 (2019-04-15)

- Implement testing actions 'changeLabel' and 'changeLocalizedEnumValueLabel' for product type

2.48 2.5.0 (2019-04-08)

- Generate the code normally created by the attrs package to improve the import time with 30%. Attrs is now also no longer a dependency
- Add py.typed placeholder file (PEP561)

- Add support for the ShoppingList service endpoint (including mock interface)
- Add support for the Reviews endpoint (including mock interface)

2.49 2.4.2 (2019-03-18)

- Tax rates ids are now correctly generated

2.50 2.4.1 (2019-02-21)

- Fix cart create function using incorrect schema
- Add update actions to carts testing backend
- Add update actions to orders testing backend
- Regenerate code based on new commercetools-api-reference definition

2.51 2.4.0 (2019-02-13)

- Refactor the code generation module to split the types.py and schemas.py file into multile submodules. This should be fully backwards compatible and from the API perspective nothing should be changed, but it makes the generated code more maintanable since we no longer have files with 40k lines.
- Use timezone aware timestamps in the mocking interface
- Rewrite the Paginator class. This is a backwards incompatible change but it makes the API much cleaner. It now also supports slicing etc.
- Add a CursorPaginator class, which uses filtering instead of offsets for pagination.

2.52 2.3.0 (2019-01-28)

- Money -> TypedMoney conversion not setting default to CENT_PRECISION
- Improve testing interface to only raise conflict error when the data is actually changed

2.53 2.2.0 (2019-01-24)

- Tax related fields now parse rate as float instead of always being 0
- Support generating code for Python 3.7 and higher
- Add limited testing service for orders

2.54 2.1.0 (2019-01-23)

- Add *commercetools.predicate.QueryPredicate()* which can be used to construct query predicate strings using Python code.
- Fix commercetools mock server

2.55 2.0.0 (2019-01-20)

- When creating the client the token url should now be a complete URL. When using environment variables it will automatically append /oauth/token if the url has no path specified. See #27
- Add a changelog :-)
- Add support for the product discounts api endpoints

2.56 1.5.0 (2018-12-11)

- **Add ability to run the mock server standalone using:** *python -m commercetools.testing.server*
- Add paginator class
- Add support for delete operations in the mock services
- Support oauth2 authentication via basic auth and querystring in the mock service
- Properly support predicate filtering in the mock endpoints using a custom query parser.

2.57 1.4.0 (2018-11-16)

- Add HTTP retry logic for status codes 502, 503, 504 (retry 3 times).
- Add Client.products.upload_image() endpoint
- Add Inventory endpoints

2.58 1.3.0 (2018-11-05)

- Add documentation via read the docs

2.59 1.2.0 (2018-11-03)

- Add support for custom objects (including mock endpoints)
- Fix handling of discriminator fields

2.60 1.1.0 (2018-11-02)

- Add support for channels endpoint
- Fix packaging issues
- Workaround a bug in marshmallow_enum ini combination with many=True

2.61 1.0.0 (2018-11-02)

This is the initial opensource release of the Commercetools Python SDK.

- Add travis support in combination with tox
- Properly support passing list and scalar values to where, sort and expand

2.62 0.6.0 (2018-10-26)

- Fix project projections mock service
- Store product type in the product mock service

2.63 0.5.1 (2018-10-25)

- No longer require full token url, instead append /oauth/token automatically

2.64 0.5.0 (2018-10-24)

- Sync with latest commercetools api reference
- Update product projections mock endpoint to use same products as product mock endpoint.
- Update payment and transaction endpoints
- **Add ability to run the mock server standalone using:** *python -m commercetools.testing.server*

2.65 0.4.1 (2018-10-19)

- Add mock endpoint for product projections

2.66 0.4.0 (2018-10-11)

- Add support for expand parameter in product projections endpoint
- Add support for product types
- Sync with latest commercetools api reference

2.67 0.3.1 (2018-10-09)

- Use the cached oauth2 token if present

2.68 0.3.0 (2018-10-09)

- Implement ability to customize the oauth2 token saver and implement a proper default mechanism which uses threading.local()

2.69 0.2.0 (2018-10-05)

- Add categories mock endpoint for testing
- Update product and category unittests

2.70 0.1.0 (2018-10-02)

Initial release of a code generated Python SDK.

Python Module Index

C

Index

F

G

I

O