https://cloudblue.com

Usage Reporting for Pay-as-you-go Items

Auto-generated at June 2, 2025

Overview

Enable the Pay-As-You-Go capability to implement the corresponding system and report subscription usage data to your Providers via the CloudBlue Connect platform.

The following instructions showcase how to configure the Processor and implement the **QT** (quantity schema) usage reporting for your specified pay-as-you-go (PAYG) items. The quantity schema enables Vendors to report quantity of consumed items. In general, Vendors generate these usage reports and submit them to Providers via the **Usage** module of the Connect platform once a month. However, Vendors can also deploy the Processor to generate such reports from their systems.

The Vendors API allows generating the QT-based usage reports that includes configured PAYG items. Therefore, the Processor can provide monthly reports for each of your products and for each of your distribution contracts. Furthermore, the Processor can submit these reports for corresponding Providers.

Prerequisites

Before following the provided use case, make sure that all of these prerequisites are met:

- The Pay-as-you-go capability QT reporting schema and are enabled.
- Your product with PAYG items is defined on the Connect platform.
- The Processor SDK Template is successfully deployed.
- Your deployed Processor is properly configured.
- Vendor API configuration is presented.
- At least one subscription (for a product with PAYG items) that was in the *Active* state last month.
- Subscription usage data that is available by using Vendor API.

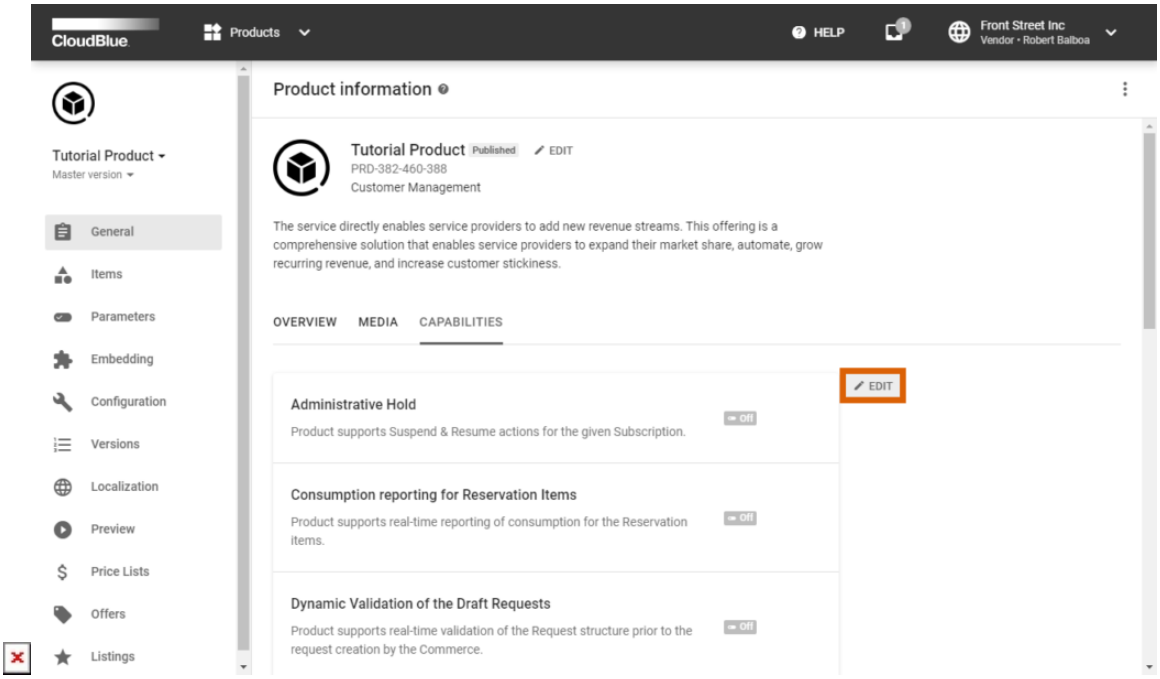
Usage Report Files

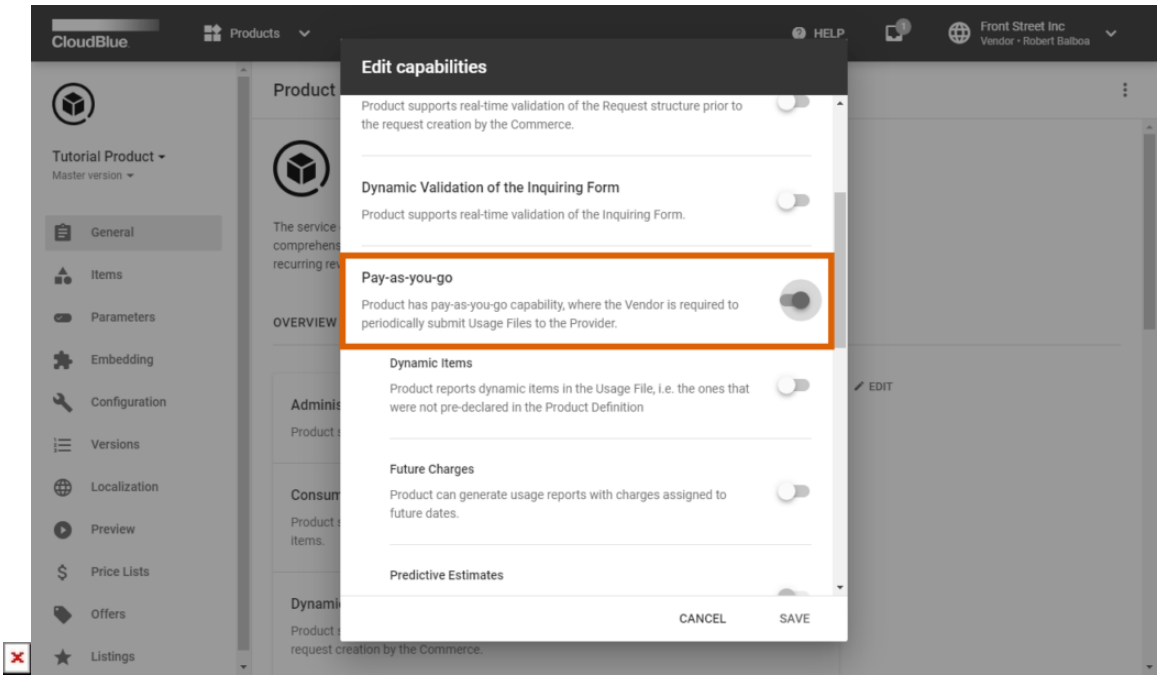
Usage report files generated via the Processor will be stored within the Usage management module of the Connect platform. This module allows submitting usage report files to Providers. Once a Provider receives a generated usage report file, this Provider can accept or reject it. Follow the instructions below to enable usage report files creation on the CloudBlue Connect platform.

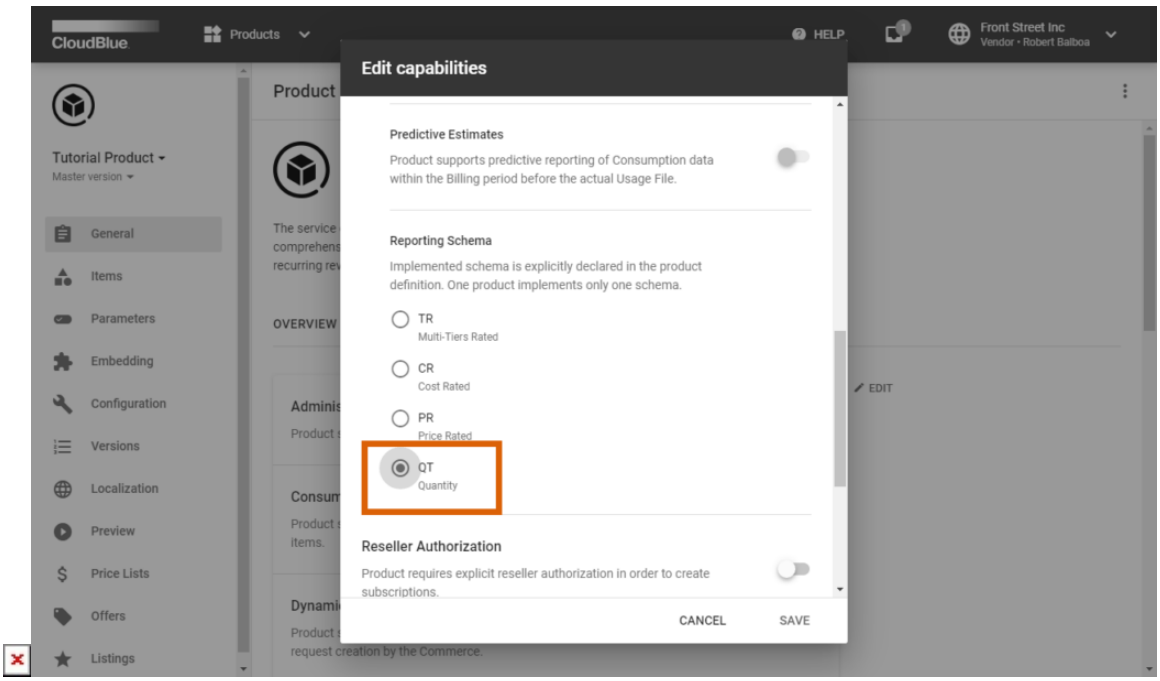
Edit Capabilities

Before generating a usage report file, make sure that the **Pay-as-you-go** capability and your selected **reporting schema** are enabled.

Proceed to the **Capability** tab from the **General** section of your product profile page. Thereafter, click the **Edit** button to switch on required capability and reporting schema.







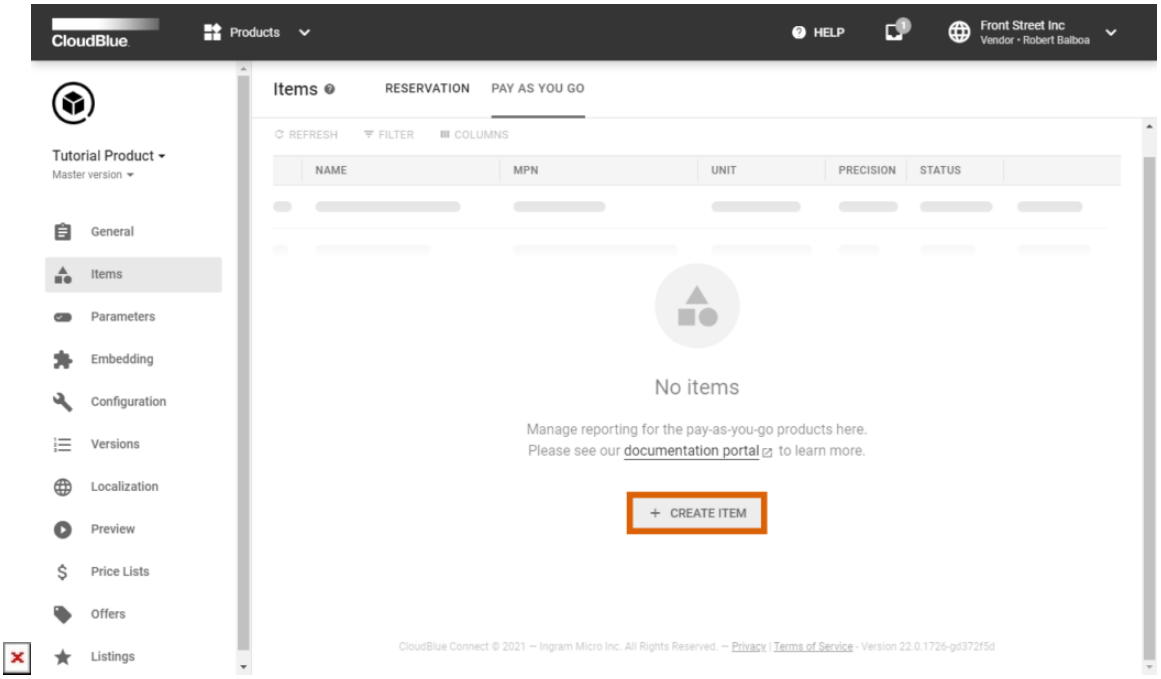
Note that the **QT reporting schema** should be enabled.

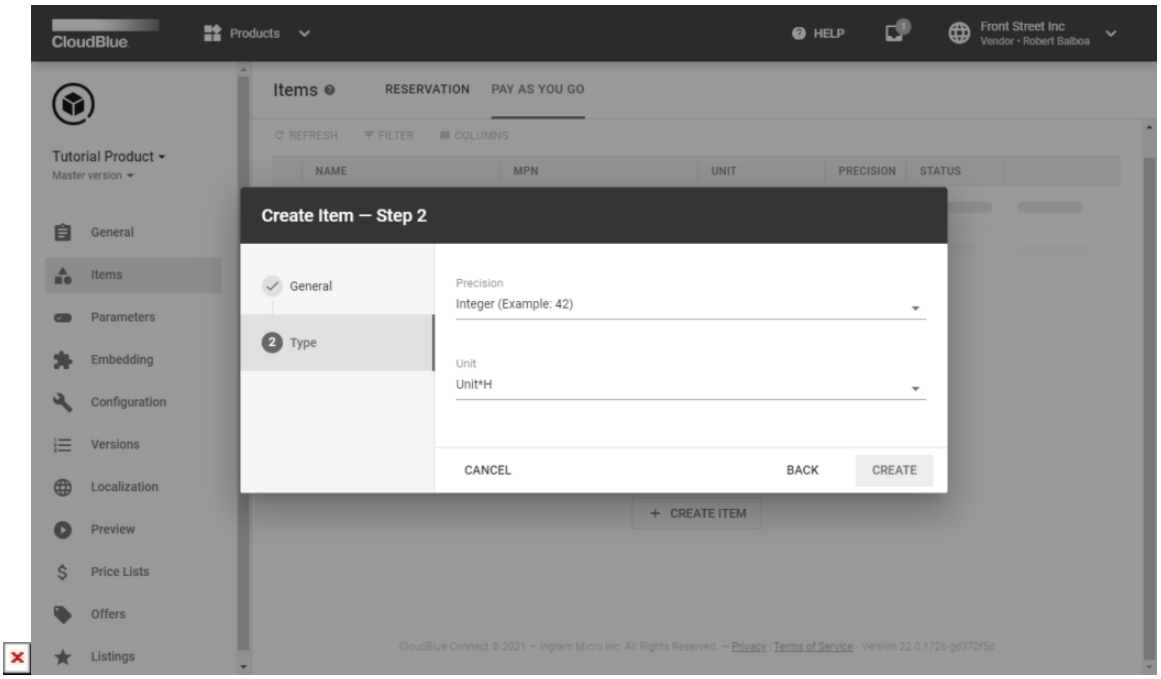
Pay-as-you-go Items

Next, this scenario requires to create and configure PAYG items.

Proceed to the **Items** section from your product profile page and switch to the **Pay As You Go** tab.

Thereafter, click the **Create Item** button to specify your PAYG items on the Connect platform.





Integer Precision

The quantity reporting schema is usually requires the **Integer** precision during the PAYG items creation.

Refer for the Pay-as-you-go Items documentation for more details.

Select a Subscription

Once your PAYG items are successfully specified, select a subscription that was in the *Active* state last month and that allows fetching quantity data via Vendor API.

Learn more about usage report files and their creation process on the CloudBlue Connect platform from to the Usage management module documentation.

Code Snippet

Create a script that generates and fills out usage report files via the Vendor API calls.

```
class Usage:
    def __init__(self, client: ConnectClient):
        self.client = client

    # This method loads contracts and subscriptions to create and fill out a usage report file.
    def process_usage(self):
        usage_data = Usage._get_usage_records()

        # Customize the Vendor API response.
        if Usage._count_usage_records(usage_data) == 0:
            # In case there is no data to report.
            return

        contracts = self._get_contracts()
        counter = 0
        # The following provides usage data for each contract.
        for contract in contracts:
            counter += self._report_contract_usage(contract, usage_data)

        if counter != Usage._count_usage_records(usage_data):
            # In case not all subscriptions with data included in your report.
            raise ValueError()

    def _report_contract_usage(self, contract, usage_data):
        # The following filters the subscriptions with the usage data.
        # For the preview environment, all the subscription belongs to contract CRD-00000-00000-00000
        subscriptions = self._get_subscriptions(contract, self._get_subscription_filter(usage_data))
        product_id = None
        record_data = []
```

```
        counter = 0
        # Locates the subscriptions with usage data for each contract.
        # It is necessary to create a Usage instance for each product.
        for subscription in subscriptions:
            counter = counter + 1
            if product_id is None:
                product_id = subscription['product']['id']
                self._validate_ppu_schema(product_id)
            if product_id != subscription['product']['id']:
                self._create_usage(contract, product_id, record_data)
                product_id = subscription['product']['id']
                self._validate_ppu_schema(product_id)
                record_data.clear()
            else:
                # The following fills the records with subscription usage data.
                record_data.append(Usage._get_usage_data(subscription, usage_data))
        if product_id is not None:
            self._create_usage(contract, product_id, record_data)
        return counter

    def _get_contracts(self):
        # The following loads distribution contracts to find subscriptions with data and generate reports.
        query = R()
        query &= R().type.oneof(['distribution'])
        query &= R().status.oneof(['active'])
        contracts = self.client.collection("contracts").filter(query)
        return contracts

    def _get_subscriptions(self, contract, subs_id_filter):
        query = R()
        query &= R().contract.id.oneof([contract['id']])
        query &= R().id.oneof([subs_id_filter])
        subscriptions = self.client.collection("assets").filter(query)
        subscriptions.order_by('product.id')
        return subscriptions
...
def _create_usage(self, contract, product_id, record_data):
    # Type: (Any, str,  [UsageData]) -> None
    # Add a usage file for your contract and your product.
    usage_file_id = self._create_usage_file(contract, product_id)
    # Usage report files provides records with PAYG items usage details.
    usage_excel_path = Usage.UsageFileExcelCreator().create_usage_excel(record_data)
    # Upload this file to the Connect platform.
    self._upload_usage(usage_file_id, usage_excel_path)
    # Submit this file to your Provider.
    self._submit_usage(usage_file_id)
```

```
# The following generates monthly reports on the platform for a particular contract and product.
def _create_usage_file(self, contract, product_id):
    # Type: (Any, str) -> str
    # The report file ID uses the following format: UF-yyyy-mm-####-####
    contract_id = contract['id']
    currency = self.client.collection('marketplaces')[contract['marketplace']['id']].get()['currency']
    date_to = datetime(datetime.today().year, month=datetime.today().month - 1,
                       day=monthrange(datetime.today().year, datetime.today().month - 1)[1])
    date_from = datetime(datetime.today().year, month=datetime.today().month - 1, day=1)
    name = date_from.strftime("%d-%m-%Y") + " - " + date_to.strftime("%d-%m-%Y")
    # Customize to assign the external file id as your need.
    external_file_id = contract_id + " " + product_id + " " + datetime.today().strftime("%B")

    payload = {
        "name": name,
        "description": "Usage " + datetime.today().strftime("%B"),
        "period": {
            "from": date_from.strftime("%Y-%m-%d"),
            "to": date_to.strftime("%Y-%m-%d")
        },
        "currency": currency,
        "product": {
            "id": product_id
        },
        "contract": {
            "id": contract_id
        },
        "external_id": external_file_id
    }
    file = self.client.ns('usage').files.create(payload=payload)
    return file['id']

def _upload_usage(self, usage_file_id, usage_file_path):
    # Type: (str, str) -> None
    # Uploads a generated Excel file to the Connect platform.
    file_data = open(usage_file_path, 'rb').read()
    file_name = ntpath.basename(usage_file_path)

    payload = {
        "usage_file": (file_name, file_data)
    }
    self.client.ns('usage').files[usage_file_id].action('upload').post(files=payload)
# The following submits the usage report to Providers.
def _submit_usage(self, usage_file_id):
    # Type: (str) -> None
    self.client.ns('usage').files[usage_file_id].action('submit').post()
```

Summary

As a result, a usage report file is generated for each product and each distribution contract. Note that the system will submit your generated usage report file to your Provider and will assign the *Pending* status to this report.