Using the SDK to export products to bepado

support@bepado.com

26th August 2014

Contents

Tutorial: Using the SDK to export products to bepado

Exporting products to bepado is the most simple step that you can implement. If you are just exporting products to bepado the shop can use the following very basic bepado functionality:

- Window-Shopping Have your products listed on bepado.com - redirecting to your own shop.
- Cloud-Search Appear in cloud search results of other shops when they didn't have a matching product for a customer.

This step is also a requirement for more advanced functionality:

- Sell products through other shops

As a prerequisite you have to go through the "Setting up the bepado SDK" tutorial.

Overview

To synchronize products from your shop with the bepado platform, bepado fetches only the changesets of products in very short intervals. Instead of fetching the complete feed of all your products once every day, bepado will only fetch the changes that happened between now and the last synchronization. bepado will also only fetch a limited amount of changes at every interval, so that your shop system is not continously under high load because of bepado.

Synchronizing only changes allows bepado to have much more recent information about the availiblity of products. In combination with a check of availability during a transaction, this allows bepado to guarante near-realtime availablity in all shops selling your products.

To allow this approach to work, you as a plugin developer have to notify the bepado SDK of changes to exported products. You can programatically notify the SDK of new products that should be exported to bepado, update products when the price or availability changed or delete products from bepado, when they should not be sold anymore.

Changesets are saved locally in a MySQL table bepado_change that contains a representation of all changes to your exported products. bepado will cleanup the table and remove changes that were already fetched.

Implement ProductFromShop#getProducts()

The only method that you need to implement for this functionality is the method getProducts() on the Bepado\SDK\ProductFromShop interface.

When this method is triggered from the SDK, it is passed a list of one or more IDs of your local product catalog. To implement this method, you must fetch the products from the database and convert them into an array of Bepado\SDK\Struct\Product instances.

It is as simple as that, but you should closely inspect the Product class to see what information is required, optional and what the format is of each of the values. The class Bepado\SDK\Struct\Product has a very complete Docblock documentation, however lets go over all the important fields here:

- sourceId should contain the ID of your product in the shop or ERP system. This value should not change over the lifetime of the product.
- ean contains the European Article number (EAN). This field is optional.
- url contains an absolute URL to the detail page of your article. This url will be redirected to from the CloudSearch and Window-Shopping functionalities. Products that do not contain this value will *NOT* be visible, so for the purpose of this tutorial its vital that the information is available.
- title is the name of the product. This field is required.
- shortDescription and longDescription are descriptions of the product. These fields are required.
- vendor contains the name of the producing Vendor of this article.
- vat contains the Value-Added-Tax that is added to this product in the shop owners home country. This is important for differentiating reduced vs full VAT priced articles in countries such as Germany where the VAT is either 7% or 19%.
- price contains the net price of the product that customers pay in your shop to buy this product. This is the price that is listed on the article detail page.
- purchasePrice contains the reduced net price that other shops pay you to resell the product to their customers.
- deliveryDate contains a unix timestamp in the future, when the article is not released yet.
- availability contains the number of items in stock of this product.
- images contains an array of URLs to pictures of this product. The first image in this list is considered to be the main image.
- categories - A list of categories this product is enlisted in. Categories here may only be valid entries from the en_US Google Category taxonomy listed here: http://www.google.com/basepages/producttype/taxono US.txt
- deliveryWorkDays - A maximium number of delivery days that this product is shipped to the country of the shop owner.

Lets assume we can retrieve all this information from our shop system, then an implementation might look like this:

```php
<?php
use Bepado\SDK\ProductFromShop;
use Bepado\SDK\Struct;

class MyProductFromShop implements ProductFromShop
{
    public function getProducts(array $ids)
    {
        $products = Shopsystem::getProducts($ids);

        $sdkProducts = array();

        foreach ($products as $product) {
            $sdkProducts[] = $this->convertToSdkProduct($product);
        }

        return $sdkProducts;
    }

    private function convertToSdkProduct(ShopProduct $product)
    {
        $sdkProduct = new \Bepado\SDK\Product();
        $sdkProduct->sourceId = $product->getId();
        $sdkProduct->title = $product->getTitle();
        // ....

        return $sdkProduct;
    }

    // other methods...
}
```

To test this method, you can write an automatic test using PHPUnit if that is easily possible with your shopsystem and just call $productFromShop->getProducts() with ids of products in your demo data.

Another way to test this is, you can write an admin page in your shop that display all the information exported to bepado by calling $productFromShop->getProducts() with some IDs given.

Mark products for export to bepado

Now that we can convert shop products to bepado products, we can export them. Exporting products to bepado always happens explicitly, that means you can choose which products to export to bepado and which ones not to export.

To export a product, you need to add a button, link or mass-export functionality in your admin area. Add a new POST request handler in your system that marks product for export:

```php
$sdk = createBepadoSDK();

foreach ($_POST['ids'] as $id) {
    $sdk->recordInsert($id);
}
```

This is the first step for exporting products. You should try to export some of your products now. You will probably recieve exceptions because you havent converted all the information of the products correctly.

The following steps are necessary to make this code more robust:

1. Check if the product was exported before and use recordUpdate instead.
2. Catch Exceptions and allow users to correct the errors.

A solution might look like this:

```
$sdk = createBepadoSDK();

foreach ($_POST['ids'] as $id) {
    $sql = 'SELECT count(*) FROM myplugin_bepado_status WHERE id = ?';
    $exported = $connection->fetchColumn($sql, array($id)) > 0;

    try {
        if ($exported) {
            $sdk->recordUpdate($id);
        } else {
            $sdk->recordInsert($id);
        }
    } catch (RuntimeException $e) {
        $sql = 'UPDATE myplugin_bepado_status SET error = ? WHERE id = ?';
        $connection->executeUpdate($sql, array($e->getMessage(), $id));
    }
}
```

It contains a table myplugin_bepado_status that you should create in your bepado plugin to keep track of all the products that you have exported and their status.

Update products when they change

You should hook into your shop system and update products when they change. This is expecially important for changes of the price, purchasePrice and availability fields.

You can delay changes to other products for a longer period, but these three should lead to updates as soon as possible.

See a simplified example how we do this in the Magento plugin for bepado:

```
public function onProductChanged($observer)
{
    $product = $observer->getEvent()->getProduct();

    if (!$product->hasDataChanges()) {
        return;
    }

    $registry = Mage::getSingleton('qafoolabs_bepado/sdkRegistry');
    $sdk = $registry->getSDK();
    $sdk->recordUpdate($product->getId());
}
```

Delete products from bepado

As a last step, you should implement the possibility to delete products from bepado using the $sdk->recordDelete($id) function.

Summary

In this tutorial we implemented one part of the bepado functionality to act as a supplier of products, for now only to be visible on bepado.com and in the CloudSearch.

As a next step you will need to implement accepting orders for these products programatically, so that other companies cann sell your products on their shops.