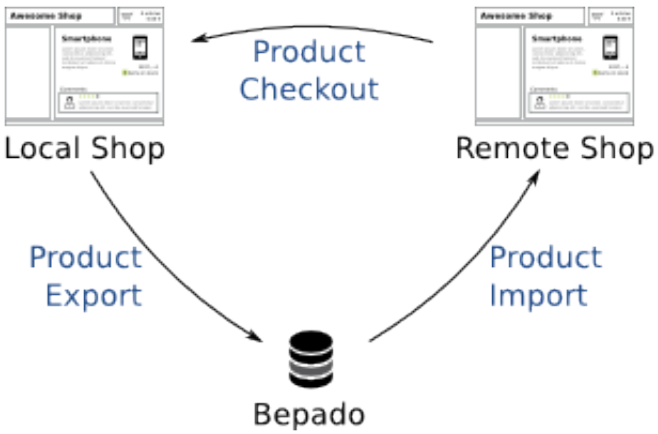
Bepado SDK API Documentation

This document describes how to use the SDK in a shop, or write an extension / module for a shop system using the API.

Basic Concepts

The basic idea is, that a shop exports a set of products, which are then synchronized with Bepado. Other shops may import those products and then customers in those shops may buy these products. This results in three basic operations:



- Product Export

  A set of exported products is synchronized with Bepado

- Product Import

  A set of products, which the shop owner configured in Bepado are made available in the local shop.

- Product Checkout

  A product is bought in a remote shop, and the transaction is completed in the local shop.

Each shop can act in both roles of course, or just in one of those roles. Product Export and Product Import are both optional. Most common case is, that a shop acts in both roles, though.

Using The SDK

Using the SDK in your shop requires you to do three things:

1. Construct the SDK

   When creating a new instance of the SDK to use in your shop, it requires five parameters. Find more about these parameters below.

2. Create a service endpoint URL for the SDK.
3. Call methods on the SDK depending on the actions inside your shop.

Construct the SDK

The SDK receives five parameters. The parameters in more detail:

- The `$apiKey`

  You get the API key for your shop instance after registering with the Bepado website. If you develop a SDK for a certain e-commerce system you probably want to provide the shop owner with a simple

user interface to configure the API key to use and pass this one to the SDK.

The `$apiKey` will be validated on the first usage of the SDK.

- The `$apiEndpointUrl`

  The URL MUST be an absolute URL under which your shop can be reached from Bepado. You extension must handle requests to this URL and pass the data on to the SDK. More on that below.

- The `Gateway`

  The `Gateway` is used by the SDK to store various data. There is a default implementation using the PHP `mysqli` extension, which you might want to use. If your shop does not use `mysqli` or you must support additional databases you should implement a custom `Gateway` for this. If you do this, we would love to this contributed back into the SDK.

- The `ProductToShop` interface

  This interface is used when products are imported into your shop. You must implement this interface to work with your shop. The interface is fairly simple and contains two methods. One to create or update a remote product, and one to remove a remote product again.

```
<?php
/**
 * This file is part of the Bepado SDK Component.
 *
 * @version $Revision$
 */

namespace Bepado\SDK;

/**
 * Interface for product importers
 *
 * Implement this interface with shop specific details to update products in
 * your shop database, which originate from bepado.
 *
 * @version $Revision$
 * @api
 */
interface ProductToShop
{
    /**
     * Import or update given product
     *
     * Store product in your shop database as an external product. The
     * associated sourceId
     *
     * @param Struct\Product $product
     */
    public function insertOrUpdate(Struct\Product $product);

    /**
     * Delete product with given shopId and sourceId.
     *
     * Only the combination of both identifies a product uniquely. Do NOT
     * delete products just by their sourceId.
     *
     * You might receive delete requests for products, which are not available
     * in your shop. Just ignore them.
```

```
     *
     * @param string $shopId
     * @param string $sourceId
     * @return void
     */
    public function delete($shopId, $sourceId);
}
```

- The `ProductFromShop` interface

  This interface is used when products are exported from your shop. You must implement this interface to work with your shop. The interface contains two methods to verify the products, which the shop currently exports, and two methods to perform an actual buy.

```
<?php
/**
 * This file is part of the Bepado SDK Component.
 *
 * @version $Revision$
 */

namespace Bepado\SDK;

/**
 * Interface for product providers
 *
 * @version $Revision$
 * @api
 */
interface ProductFromShop
{
    /**
     * Get product data
     *
     * Get product data for all the product IDs specified in the given string
     * array.
     *
     * @param string[] $ids
     * @return Struct\Product[]
     */
    public function getProducts(array $ids);

    /**
     * Get all IDs of all exported products
     *
     * @return string[]
     */
    public function getExportedProductIDs();

    /**
     * Reserve a product in shop for purchase
     *
     * @param Struct\Order $order
     * @return void
     * @throws \Exception Abort reservation by throwing an exception here.
     */
```

```
    public function reserve(Struct\Order $order);

    /**
     * Buy products mentioned in order
     *
     * Should return the internal order ID.
     *
     * @param Struct\Order $order
     * @return string
     *
     * @throws \Exception Abort buy by throwing an exception,
     *                    but only in very important cases.
     *                    Do validation in {@see reserve} instead.
     */
    public function buy(Struct\Order $order);
}
```

Creating a Service Endpoint

The service endpoint URL is used by Bepado and other shops to interact with your shop. All other instances execute `POST` requests against this URL. The URL itself does not matter, but the absolute URL must be provided to the SDK using the `$apiEndpointUrl` constructor parameter.

When receiving a request to this URL the XML body of the request must be dispatched to the `handle()` method on the SDK object. The `handle()` method will then again return a XML string, which should be echo'd. You must not echo anything but the raw XML returned by the method.

A simple PHP file, which does this could look as simple as:

```
<?php

$sdk = new Bepado\SDK(
    'my-api-key',
    'http://example.com/endpoint',
    new My\Gateway( /* … */ ),
    new My\ProductToShop(),
    new My\ProductFromShop()
);

echo $sdk->handle(file_get_contents('php://input'));

?>
```

Calling the SDK

The SDK class has a set of methods, which provides you with the means to interact with Bepado. The available methods can be related to the following tasks:

1. Registering products for export
2. Checkout of remote products
3. Bepado Cloud Search

On top of that there are two helper methods:

1. `verifySDK()` checks if the API key is valid and registers the API endpoint URL with Bepado.

2. `getCategories()` provides you with a list of valid Bepado categories to associate your products with.

The three tasks and the according methods are described in more detail in the following sections.

Register Products for Export

For product export Bepado must know about the products, which your shop wants to make available and must be informed about all changes to those products.

The first step you need to implement, which is entirely dependent on the shop you are targeting is a method to configure the set of exported products. This is independent from the SDK itself.

Once the list list of exported products is configurable, the SDK must be informed about the products (and later of changes to the listed products, or changes to the list). There are two ways to inform the SDK about this:

Update notifications

The SDK is called for every update to a product or the list. This method does require far less processing power in the SDK, but, depending on your shop, might be hard to implement.

Using this method the SDK must also be informed about all changes, which might be applied by third party tools, like ERP systems etc.

Use the `recordInsert()`, `recordUpdate()` and `recordDelete` methods on the SDK object for this. Just call the respective method every time a change to a product or the product list occurs.

Change evaluation

The SDK can itself verify which changes were made to the list or products. For large amounts of products this check might consume lots of processing power. It usually is far easier to implement with existing shop systems, though.

Simply call the `recreateChangesFeed()` method for this, and the SDK will do everything else. But remember: This way costs far more processing power.

You should either call this method every time the currently exported products are requested by Bepado using the service endpoint, or every time any change happens to the product database.

Checkout of Remote Products

If, during checkout, the shopping cart contains products from a remote shop the SDK provides you with a set of methods to verify the integrity of the remote products and perform the actual checkout in the remote shops.

Check Products

The method `checkProducts()` allows you to verify that the remote products still have the same price and availability, as stored in your local database. You should call this method when you want to verify an order.

The method receives an order struct, which should contain all products, which should be verified. The SDK will verify that the order struct is valid.

Reserve Products

The method `reserveProducts()` should be called when you want to reserve products in the remote shop(s). You must call this method as a part of the checkout process. If relevant product details changed, compared to the data your shop has stored locally, the method will return a message struct. this should be presented to the user. Examples for this are price and availability changes of products.

If the reservation succeeded a set of hashes will be returned, which should be stored in the user session. Those hashes will be used to finalize the checkout process.

Checkout Products

The final step to buy products is calling the `checkout()` method. This step receives a reservation struct, as returned by the `reserveProducts()` method. Usually the method will just return `true` and the buy process can be considered completed.

If some problem occured while finalizing the checkout, the method will return a message struct. Those messages should again be presented to the user, so the user can take appropriate action. Alternatively you can still present the user a success message and let your support team handle the issue.

Cloud Search

The Bepado Cloud Search enables you to search for products in foreign shops. Just call the `search()` method with an appropriate search struct, containing the search terms and optionally additional search configuration. The method will return a search result struct, which you can present as a search result on your website. The structs should be self-documenting.