Programmers Guide

eBay Acellerator Toolkit for PHP

Shopping-API Edition

EbatNs_Shopping 1.0

© 2007 Intradesys Limited

1 Introduction

The eBay Acellerator Toolkit New Schema (EbatNs) is the next step in programming the eBay API with PHP. It is the suceessor of the eBay Acellerator Toolkit (ebat) for the legacy schema that already exists since 2003. The toolkit gives you a kind of SDK for usage in PHP application on the webserver and/or the command line.

As eBay decided to publish a second API (aka Shopping-API) this API is now supported by a special edition of the “eBay Acellerator Toolkit for PHP – Shopping API edition” (EbatNs_SAE). This API has a limited functionality relating to the former API (aka Trading-API).

The former EbatNs will continue to exist with no changes anyway we might refer to EbatNs_TAE to separate both editions.

2 Installation

To install the EbatNs_SAE you will need a fully functional PHP installation on your system. We have tested with PHP4 and PHP5 – so you should be fine with any version higher 4.

Please first download the package and unzip it. The package will be placed in a folder named “EbatNs_Shopping_1_0_XXX_0” – where xxx is the current version of the API. Per default you will find a sample-folder within the installed package.

2.1.1 Needed PHP Installation on the Server

As the Shopping API does not use HTTPS but nonsecure HTTP you will not need any further installation.

Per default you can use a cURL installation on your server to handle the underlaying HTTP-transport.

As the Shopping-API does not use a secure connection (HTTPS) you can use also a provided HTTP-Transport-Module that does NOT use cURL. If the file is include the switch is done transparently in the background.

To use the NON-cURL transport just include the file “`EbatNs_HttpTransport.php`” BEFORE you include the Proxy-File:

```
require_once '../EbatNs_HttpTransport.php';
require_once '../EbatNs_ServiceProxyShopping.php';
…
```

3 First Steps

3.1.1 Join the eBay Developer-Program

To use any of the eBay-API-Calls you will have to have an account in the eBay developer program.
https://developer.ebay.com/join/Default.aspx

After registration you can “Generate Keys”. For the using the Shopping API you need only the so called “APP-ID”. There is NO Sandbox-Access for the Shopping-API so you preferable generate Production Keys only. Using a Sandbox-App-ID will result to an API error (Error #1.20: Application ID invalid.) !

You might use an already existing APP-ID if you had previously registered for the Trading-API.

Be aware that the call limits for the Shopping-API are based on number of requests at a certain IP-address per day. Details could be found here: http://developer.ebay.com/products/shopping/Default.aspx (Access and Rate Limiting)

3.1.2 Create a basic configuration

While programming with the toolkit you will always setup first a EbatNs_Session-Object.
For using the Shopping-API you will need only the APP-ID (see above) and the Site-ID (e.g. 0 for eBay.com, 77 for eBay.de, etc.).

```
$session = new EbatNs_Session();
$session->setAppId($app_id);
$session->setSiteId($site_id);
```

Note: For users of the Trading API. If you already have a .config File for the Trading-API you can just reuse it for the Shopping-API also. Both editions are based same functionality regarding the EbatNs_Session-Object.

3.1.3 Samples

The first sample should run now, try „index.php“. Navigate to your webserver to the sample-folder. You will have to enter your APP-ID and a SITE-ID.

4 How to use the EbatNs Trading-API Edition and Shopping-API Edition together

This is only for users who would like to use both Editions together.

- Parallel Operation means installing both editions (Trading-API and Shopping-API) to single installation folder and use both Editions in parallel in a single PHP script. Of course you can also separate installation and script environments by altering your include path.
- Parallel Operation is only possible with EbatNs Trading-API Edition Version 525
- You should NEVER try to run different version (Compatibility Level) of both editions together. As both editions will share certain files, this might result to unpredictable results.

To install both editions together you just need to first install the Trading-API edition and then install the Shopping-API. Now you can copy over (almost) all files from the Shopping-API folder to the Trading-API folder except these files:

- AbstractRequestType.php
- AbstractResponseType.php

The 2 files are shared in both editions, anyway the Shopping-API can work with the version you find in the Trading-API but not vice-versa.

Using both editions can share the same EbatNs_Session Object. So also using a .config-file to simplify the session-setup can be used with the Shopping-API. All other advanced features of the Trading-API (DB-Handlers, Extensions for the AttributeSets, Batch-Calls etc.) can now also be used with the Shopping-API.

5 Basic Operations

To issue a call to the eBay API you will need a Session. Then create a ServiceProxyShopping with the Session. The parameters of a call are always given in a call-specific Request-Object. The ServiceProxyShopping will have a PHP-Method for all known API-call that takes the Request as a parameter and returns the call-specific Response-Object.

5.1.1 Proxy, Session

So first lets create a Session and a ServiceProxy.

```
require_once '../EbatNs_HttpTransport.php';
require_once '../EbatNs_ServiceProxyShopping.php';
$session = new EbatNs_Session();
$session->setAppId($app_id);
$session->setSiteId($site_id);
$cs = new EbatNs_ServiceProxyShopping($session);
```

5.1.2 Request, Call, Response

Having the ServiceProxyShopping you can setup your Request-Object. To not overload the memory with preloading files, you have to include the needed type-file for your own. But this is easy: If you know the Name of the Call to issue just append "…RequestType.php".

So let's try with a FindItems-Call. The responding Request-Type would be "FindItemsRequestType.php"

```
require_once '../FindItemsRequestType.php';
$req = new FindItemsRequestType();
$req->setQueryKeywords($_REQUEST['query']);
```

The Request-Type might (normally) need various data to be set. Here as we want to do a FindItems-operation we only need to set the QueryKeywords. Within the EbatNs you will find "setter"-methods for all parameters you can use.

Next you can issue the call:

```
$res = $cs->FindItems($req);
```

Easy doing you will get back the response with the Item you are interested in.

Next you can check the Response and retrieve the information of interest. In this case we can just print out the item.

```
echo "<pre>";
print_r($res);
```

Also the Response Object will have methods to access the data. Here you will find "getter"-methods for all data.

Note: Please take care that not all data is returned by all calls. The eBay Unified Schema might define data for various types that is only returned under certain circumstances.

Ok, here is the complete script altogether:

```
require_once '../EbatNs_HttpTransport.php';
require_once '../EbatNs_ServiceProxyShopping.php';
require_once '../FindItemsRequestType.php';
$session = new EbatNs_Session();
$session->setAppId($_REQUEST['app_id']);
$session->setSiteId($_REQUEST['site_id']);

$cs = new EbatNs_ServiceProxyShopping($session);
$req = new FindItemsRequestType();
$req->setQueryKeywords($_REQUEST['query']);

$res = $cs->FindItems($req);

echo "<pre>";
print_r($res);
```

5.1.3 Handling Errors

The eBay API will respond with error information in certain situations. You might check this condition to handle these errors. Please be aware that you might get errors and warnings, whereas warning will mainly have only informational intense.

To check the success (nor errors or warnings) by check the field Ack in the Response-Object:

```
$req = new GetSingleItemRequestType();
$req->setItemID(1);
$res = $cs->GetItem($req);
if ($res->getAck() != 'Success')
echo "we got a failure";
```

Here you can also retrieve the Errors-Array and show the error you got:

```
foreach ($res->getErrors() as $error)
{
     echo    "#" . $error->getErrorCode()
           . " " . htmlentities($error->getShortMessage())
           . "/" . htmlentities($error->getLongMessage()) . "<br>";
}
```

6 Correlation to the API Documentation

The EbatNs_SAE is directly based on the eBay Shopping API and basically you will find all needed information within the online-documentation on http://develoeper.ebay.com.
The online could be read here http://developer.ebay.com/DevZone/shopping/docs/CallRef/index.html.

As all stuff could be found here we skipped a documentation of the Toolkit and you should relate to the docs above. Be sure that you relate to the documentation version of the Toolkit you are using.
For all API-calls (found under API Call Reference Guide) you will find a method in the EbatNs_ServiceProxyShopping and the request/response datatypes. All schema datatype will correlate to a PHP file that implements the datatype.
We have made minimal adoption to the way to eBay Schema datatypes are transferred to PHP objects. You can find details within the following paragraphs.

Note: We had integrated comments within the EbatNs_SAE code to let an intelligent IDE use this information and provide you with some hints while programming. You can use a documentor on this information and generate a help file (checkout http://www.phpdoc.de/).

7 Data Structure

What was basically done her is a kind of type-aware DOM parsing but without using the PHP DOM Libraries but only the “normal” XML (Expat) parser that PHP is providing as a internal module. So you will get PHP objects with member-variables, in case of arrays these variables are just PHP arrays. The underlying EbatNs_Parser will include any needed PHP class-files if needed, so the resulting object-structure should be very clean.

7.1.1 Basic Mapping

As a basic guideline all eBay Schema datatypes are mapped against single PHP files which you can find in the EbatNs_SAE installation folder. For each call you will find a file corresponding to the Request and Response datatype. The files will auto-include any needed sub-types that are used within here.

The implementation of the datatypes in PHP are based on PHP-classes. For any child-elements these classes implements a member-variable with the same name. All variables could be either accessed directly or via provided getter- and setter-methods.

If the Schema datatype is a “simple-type” we provided a member-variable which is named “value” and a setTypeValue($value) and getTypeValue() method (see EbatNs_SimpleType.php for details).

Most of the Schema datatype do not have XML-Attributes but a few do (e.g. the AmountType). Here you can use the provided methods setTypeAttribute($key, $value) and getTypeAttribute($key) to access the Attributes. Sounds very complex but in reality it is not. PHP is very “type-lax” so we thought the best way was to do it like this.

7.1.2 Special Handling of ArrayTypes

A lot of the eBay-schema-elements are returning arrays. Anyway if you look at the names of the elements you will often but not always see names like “xxxArrayType” or a plural like “someType**s**” (so with an ‘s’ at the end).

Sometimes these arrays are called just with a singularname (e.g. InternationalShippingServiceOption instead of InternationalShippingServiceOption**S** but correctly ShippingServiceOptions). Anyway maybe the eBay guys are going to change it in the future but for the moment we have to handle this.

The schema provides information if an element has a cardinality of greater than 1 (so being an array) and so we used this information. An important issue is that such an array-element is always an array.

Sound paradox but other implementations where just returning plain-value for array of size 1, the EbatNs_SAE will correctly return – in such a situation - just an array with a single element. Although you can be sure to get an array this result also to some implications (as a few elements e.g. the parent-id of a category are defined as array-types although mostly you will only get 1 element).

For such elements the getter-/setter-method are a little different. Here you will have an optional argument $index which let you access a single element of the array.

e.g.

```
$category->getParentId(0);
$ShippingDetails->setShippingServiceOptions($option, 2);
```

Leaving out the $index argument will set/get the internal array itself (so all elements). Although all of this seems easy and obvious you might trigger some problems here as the API docs does not directly points you to the cardinality (anyway it is in the WSDL and Visual Schema Reference).

7.1.3 Setter- and Getter-methods

For all Schema object the appropriate access methods are generated. The names are just prefixed by “get” and “set”.

Schema:

```
Item.ListingDuration
```

will result to:

```
$item->getListingDuration();
$item->setListingDuration($duration);
```

If an element is an array you will get extra arguments $index to access the indexed values. Beware that leaving the argument out will result to access the internal array, but a single value.

Setting array-type values:

```
$paymentMethods[] = 'PersonalCheck';
$paymentMethods[] = 'MoneyXferAccepted';
$paymentMethods[] = 'PayPal';

// setting the array as a value
$item->setPaymentMethods($paymentMethods);
```

or

```
// setting the values one by one
$item->setPaymentMethods($paymentMethods[0], 0);
$item->setPaymentMethods($paymentMethods[1], 1);
$item->setPaymentMethods($paymentMethods[2], 2);
$item->setPaymentMethods('COD', 3);
```

Accessing array-type values:

```
// getting all as an array
$array = $item->getPaymentMethods();
```

```
// getting a single value
if ( count($item->getPaymentMethods() > 4) )
     $value = $item->getPaymentMethods(4);
```

7.1.4 Handling on “empty” values

To save memory we decided to reduce all objects to the minimum. So the EbatNs_SAE will NOT contain variables for Schema elements that are not returned from the API Servers. The eBay unified schema will always return e.g. an Item as a complete ItemType, but this does not mean that all fields will be returned. In case eBay does not return a value you will not have the member variable.

Although this should not bring you in trouble you will encounter that PHP will throw a Notice if you try to access the variable (this might be also happen if you use the get-Method).

```
error_reporting(E_ALL);
…
$item = $res->getItem();
echo "ShippingTerms : " . $item->getShippingTerms() . "<br>";
```

would throw this if the item you work on does not have ShippingTerms :

Notice: Undefined property: ShippingTerms in **../ItemType.php** on line **xxx**

You should then switch off error_reporting for E_NOTICE (which is the default on PHP 4) or test the value first.

7.1.5 Data Conversion

The EbatNs will do data conversion internally. As SOAP and PHP differs in the way certain data types are coded we did some internal conversion here. Besides you would like to work in your scripts not using UTF-8 Charset Encoding (what is needed for the eBay API) but using PHP's ISO-8859-1 charset.

We placed DataConversion into a separate class called EbatNs_DataConverter. Here you will find handler method for encoding and decoding data. You might decide to re-model the DataConverter for your needs.

IMPORTANT NOTE: Currently the EbatNs_ServiceProxyShopping is using a DataConverterIso as default which will encode/decode to ISO-8859-1. So any string sended will be encoded using utf8_encode and any strings coming back will be decoded using utf8_decode. Besides this the field "Description" is always placed into a <![CDATA[ ]]> construct.

7.1.6 UTF8 and ISO8859-1

The eBay API is working with UTF8 only – in the EbatNs_SAE you can decide if you like to go with ISO8859-1 (which is the default) or work in UTF8 mode.

To change over to UTF8 you will need to adapt the way how the EbatNs_ServiceProxyShopping is created.

```
// creating a proxy with (default) Iso-Converter
$cs = new EbatNs_ServiceProxyShopping($session);
// ... the same as
$cs = new EbatNs_ ServiceProxyShopping ($session, 'EbatNs_DataConverterIso');

// creating a proxy for UTF8 (or custom converters)
$cs = new EbatNs_ ServiceProxyShopping ($session, 'EbatNs_DataConverterUtf8');
```

8 Special EbatNs-Features

8.1.1 Using a DataType-Handler

Depending on the amount and way you would like to handle the returned data you might use the “DataType-Handler”-feature of the EbatNs.

Basically you hook into the parsing process of the ResponseParser and will be notify if a certain Element was parsed. The hook is based on the Type of object that needs to be stored. Normally the parser would just attach the data within the structure (ResponseObject) and returns all data as a return of the call.

Especially if you have to handle of large amount of data (e.g. a category-download) this way would create a very large response-object. So the script would consume a lot of memory and resources. Anyway afterwards you would go and loop the response-data and handle it.

The way the DataType-handler works will make the procedure much easier and normally you can work with a much lower memory-limit.

So setup a handler you will need a callback-function (either a global function or an object’s method). You will then inform the ServiceProxy about the handler and the DataType is should work on. Within your handler you can decide weather you like to attach the parsed object to the result or not.

The function signature has to be like this :

```
function callback($type, & $data)
{
return true;
}
```

Ok so let bring it into a bigger context and hook in for ItemTypes :

```
$cs = new EbatNs_ServiceProxyShopping($session);
$cs->setHandler('SimpleItemType', 'handleItem');
$req = new GetSingleItemRequestType();
$req->setItemID(4504181935);
$res = $cs->GetItem($req);

function handleItem($type, & $data)
{
   echo "Hello Item<pre><br>";
   print_r($data);
   return false;
}
```

The function handleItem will be called on each Item retrieved. Returning false in the function will let the parser attach the Item to the response. If you return true the Item would NOT be attached. Anyway the same functionality could also be done using an object as the handler.