Shop-Script 5 API

Bold font is used to mark existing methods, normal font is used for methods to be implemented in future software updates.

For clarity, parameters accepted by a method are specified in parentheses. In real use parameters are sent via a GET or POST request. E.g., api.php/shop.category.getTree?parent_id=1287&max_level=2&...

category

shop.category.getTree ($parent_id = 0, $depth) GET; parent_id is the id of the parent category whose subtree is requested. If not specified, the entire tree is returned. Parameter $depth is used to specify the maximum number of tree levels which will be returned as an array.
shop.category.getInfo ($id) GET; returns information about a category with the specified id.
shop.category.add (*data fields*) POST; data fields specified in the key=value form. This method returns the description of a product category (array structure is described below). The required element is 'name'; array items with keys 'left_key', 'right_key', 'type', 'full_url' are ignored.
shop.category.update ($id, *data fields*) POST; updates information about the category with the specified id. If some items (e.g., 'url') are not included in the data array then the corresponding data fields are not updated.
shop.category.delete ($id) POST; deletes the category with the specified id. Products contained in the deleted category are not deleted.
shop.category.search ($hash) **GET;** hash = 'id/1', 'name/Accessories', …

All methods utilize unified array structure to exchange category information. Below is an example in the XML format (&format=xml; if the format is not specified then JSON is used by default):

```
<id>1287</id> internal category id (PK)
<left_key>2</left_key> service field containing the category
position in nested sets; left
<right_key>3</right_key> service field containing the category
position in nested sets; right
<depth>1</depth> service field containing the category position
in nested sets; depth
<parent_id>1286</parent_id> id (PK) of the parent category; 0 for
top-level categories
<name>2013 season</name> Category name
<meta_title>Popular models of 2013 season snowboards</meta_title>
<meta_keywords>snowboards,2013,burton,neversummer,buy,worldwide</
meta_keywords> category's META Keywords
<meta_description>Catalog of popular models of 2013 season
snowboards. Online ordering & delivery.</meta_description>
category's META Description

<type>0</type> 0: static (manually filled) category, 1: dynamic
(automatically filled; e.g., by price or tag) category
```

```
<conditions/> only for dynamic categories; product selection
conditions; e.g., “rating>=0&price>=100&price<=200»
<url>2013</url> category's main (editable) URL
<full_url>snowboards/2013</full_url> full URL of the category in
accordance with its hierarchy
<count>12</count> number of products in the category


<description>
<p>Online store of snowboards. Catalog of popular models of the
2012–2013 season.</p> <p>Online ordering & worldwide
delivery.</p>
 </description> full category description (HTML)
 <create_datetime>2013-06-24 07:31:07</create_datetime> category
creation date and time
 <edit_datetime>2013-06-24 07:34:00</edit_datetime> last category
editing date and time
 <filter>price,60,61</filter> list of product feature ids which
can be used to filter products in the storefront; the specified
list of parameters is used to generate the product filtering
form; if the list is empty, the product filter is not displayed
in this category
 <sort_products>total_sales DESC</sort_products> default product
sorting order; if this field is empty, products are sorted by the
order manually selected by the administrator in the backend (only
for static categories)
 <include_sub_categories>0</include_sub_categories> this field
denotes whether the current category should display products
contained in its subcategories
 <status>1</status> 1: category is visible in the storefront, 0:
category is hidden
 <route/> denotes whether a separate frontend settlement has been
selected for the current category; if this field is empty then
the category is displayed in all storefronts; valid only for
websites with multiple storefronts
```

product

shop.product.search ($hash, $offset = 0, $limit = 500) **GET; ‘id/1’, ‘tag/apple’, ‘name/burton’**
shop.product.getInfo ($id) **GET;** returns information about a product with the specified id.
shop.product.add (*data fields*) **POST;** adds a new product by saving data from fields specified in the key=value form (array structure is described below); elements 'id', 'sku_type' are ignored; 'name' and 'skus' are required elements.
shop.product.update ($id, *data fields*) **POST;** updates information about a product with the specified id.
shop.product.delete ($id) **POST;** deletes a product with the specified id.

shop.product.addToCategory ($product_id, $categoryId) POST
shop.product.addToSet ($product_id, $set_id) POST
shop.product.addTags ($product_id, array $tags) POST
shop.product.removeFromCategory ($product_id, $categoryId) POST
shop.product.removeFromSet ($product_id, $set_id) POST
shop.product.removeTags ($product_id, array $tags) POST

shop.product.setBadge ($product_id, $badge_id, $badge_html = '') POST
shop.product.removeBadge ($product_id) POST

Methods for managing complex entities (product SKUs, images, info pages, etc.) are organized in separate groups: shop.product.skus.* , shop.product.images.*, etc.

All methods use unified product information array structure. XML format example:

```
<id>97859</id> internal product id (PK)
<name>Burton Custom X</name> product name

<summary>
Professional freestyle snowboard. Best model of the 2012–2013
season.
</summary> brief product summary displayed in product lists
(e.g., inside categories and search results)
<meta_title>Buy Burton Custom X</meta_title> title of the product
page
<meta_keywords>burton,custom x,snowboard,buy</meta_keywords>
product page META keywords
<meta_description/> Mproduct page META description
<description>
<p>Professional freestyle snowboard, ready for hard riding both
in a snow park, and on a mountain slope.</p> <p>One of the best
models of the 2012–2013 season.</p>
</description> full product description (HTML)
<contact_id>1</contact_id> id of the user (contact) who has added
this product to the catalog
<create_datetime>2013-06-24 08:52:29</create_datetime> product
creation date and time
<edit_datetime>2013-06-24 09:24:52</edit_datetime> last product
modification date and time
<status>1</status> 1: product is visible in the storefront, 0:
product is hidden
<type_id>10</type_id> product's type id (shop_type)
<image_id>3984</image_id> product's default image id
<image_url>http://wa/wa-
data/public/shop/products/59/78/97859/images/3984/3984.750x0.jpg<
/image_url> absolute URL of the default product image
<sku_id>52473</sku_id> product's default SKU id
<ext>jpg</ext> format of the default product image
<url>burton-custom-x</url> main (editable) URL of the product
page
<rating>0.00</rating> average customer rating (from 0 to 5); 0:
no rating value available
<rating_count>0</rating_count> number of product ratings
<price>20000.0000</price> product's default SKU price
<compare_price>0.0000</compare_price> “compare at” price of the
product's default SKU; 0: no value specified; “compare at” price
is used to point customer's attention to the price decrease
<currency>RUB</currency> ISO3 code of the product price currency
<min_price>19800.0000</min_price> lowest product price (among all
its SKUs)
<max_price>20000.0000</max_price> highest product price (among
all its SKUs)
<tax_id>10</tax_id> id of the tax class selected for the product;
tax classes are used to automatically calculate the tax amount
during checkout and fo generation of various printable forms
<count>11</count> total in-stock amount of the product; if no
value is specified then the in-stock amount is not tracked for
this product
```

```
<cross_selling>1</cross_selling> cross-selling status of the
product; 0: disabled, 1: automatic generation of the cross-
selling list based on corresponding settings for this product
type; 2: cross-selling list is manually specified in the product
settings screen
<upselling/> similar status value for upselling product lists
<total_sales>0.0000</total_sales> total sales amount of the
product
<category_id>1287</category_id> id of the product's main
(primary) category (used for the breadcrumbs element)
<badge/> product image badge id
<sku_type>0</sku_type> product sales mode; 0: flat SKU list from
which customers can choose the desired SKU; 1: selection of
product feature values where each combination of values
unambiguously corresponds to one of product SKUs

<base_price_selectable>0.0000</base_price_selectable> for the
featue values selection mode, (value “1” above): product's base
price is applied to an SKU if its individual price is not
specified

<images>
  <image>...</image>
</images>
<skus>
  <sku>...</sku>
  ... SKU element structure is described below ...
</skus>
<categories>
  <category>...</category>
  ... list of items containing information about all categories
in which the product resides; category information array
structure is described above ...
</categories>
<features>
  <board_length/>
  <board_type/>
  <stiffness>7</stiffness>
  <weight>5.00 kg</weight>
  ...list of product's feature values... feature ids (e.g.,
board_length, board_type, etc. are editable in the product
features management screen in the backend
</features>
```

product.skus

shop.product.skus.getList ($product_id) GET
shop.product.skus.getInfo ($id) GET
shop.product.skus.add ($product_id, *data fields*) POST
shop.product.skus.update ($id, *data fields*) POST
shop.product.skus.delete ($id) POST
shop.product.skus.addAttachment ($sku_id, $url / $file) POST
shop.product.skus.removeAttachment ($sku_id) POST

Product SKU information array structure. XML format example:

```
<id>52472</id> SKU id (PK)
<product_id>97859</product_id> SKU's product id
<sku>custom-x-152</sku> text name of the SKU; optionally
specified in the backend depending on the store's individual
product accounting requirements
<sort>1</sort> defines the sort order of the SKU in the common
product SKU list
<name>152</name> SKU name displayed in the storefront; in this
example “152” denotes the snowboard length; if a product has only
one SKU the the SKU name is not displayed in the storefront
<image_id>3985</image_id> id of the product image selected for
this SKU; SKU image id is used to update the product image upon
selecting this SKU in the storefront; if no image has been
selected for the SKU then product's default image is displayed
when this SKU is selected
<price>19800</price> SKU price expressed in product's currency
<primary_price>19800.0000</primary_price> SKU price expressed in
the store's primary currency (necessary for report generation)
<purchase_price>0</purchase_price> SKU purchase price (for report
generation)
<compare_price>0</compare_price> SKU “compare at” price (to
display the price decrease in the storefront)
<count>6</count> in-stock amount of this SKU
<available>1</available> 0: SKU is not available for ordering
(e.g., temporarily), 1: SKU is available for ordering
<dimension_id/>
<file_name/>
<file_size>0</file_size>
<file_description/>
<virtual>0</virtual> “virtual” SKU status; 0: SKU was manually
added in the product-editing screen; 1: SKU was automatically
generated in the product feature values selling mode; for
example, if 4 size values “S”, “M”, “L”, “XL” and 2 color values
“red” and “green” are selected, then 4x2 = 8 SKUs are
automatically generated; automatically generated SKUs are
considered “virtual SKUs” for convenience of their management in
the backend; once a virtual SKU has been manually edited by an
administrator, such SKU changes its status from virtual to real
<stock/> if in-stock values for multiple stocks have been
specified then this field contains SKU's in-stock information for
each stock
```

product.images

shop.product.images.getList ($product_id) GET
shop.product.images.getInfo ($id) GET
shop.product.images.add ($product_id, $url / $file) POST
shop.product.images.delete ($id) POST
shop.product.images.getThumbUrl ($id, $size) GET
shop.product.images.getFullsizeUrl ($ɯd) GET
shop.product.images.rotate ($id, $clockwise = true) POST
shop.product.images.applyFilter ($id, $filter) POST

Product image array structure. XML format example:

```
<id>3984</id>
```

```
<product_id>97859</product_id>
<upload_datetime>2013-06-24 08:53:00</upload_datetime>
<edit_datetime/>
<description/>
<sort>0</sort>
<width>750</width> original image width (px)
<height>1212</height> height (px)
<size>461446</size> file size in bytes
<original_filename>222.jpg</original_filename> original image
file name
<ext>jpg</ext>
<badge_type/>
<badge_code/>
<edit_datetime_ts/>
<url_thumb>
http://wa/wa-
data/public/shop/products/59/78/97859/images/3984/3984.200x0.jpg
</url_thumb>
```

product.pages

shop.product.pages.getList ($product_id) GET
shop.product.pages.getInfo ($page_id) GET
shop.product.pages.getContent ($id) GET
shop.product.pages.add ($product_id, data fields) POST
shop.product.pages.update ($id, data fields) POST
shop.product.pages.delete ($id) POST

product.reviews

shop.product.reviews.getTree ($product_id, $review_id = 0) GET
shop.product.reviews.getInfo ($id) GET
shop.product.reviews.add ($product_id, data fields) POST
shop.product.reviews.update ($id, data fields) POST
shop.product.reviews.delete ($id) POST

product.features (product feature values)

shop.product.features.getList ($product_id) GET
shop.product.features.getValue ($feature_id) GET
shop.product.features.add ($product_id, $feature_id, data fields) POST
shop.product.features.update ($product_id, $feature_id, data fields) POST
shop.product.features.delete ($product_id, $feature_id) POST

type

shop.type.getList ( )
shop.type.getInfo ($type_id) GET
shop.type.add (data fields) POST
shop.type.update ($type_id, data fields) POST
shop.type.delete ($type_id) POST

feature (general product feature management)

shop.feature.getList ($type_id = null) GET
shop.feature.getInfo ($feature_id) GET
shop.feature.add (data fields) POST
shop.feature.update ($feature_id, data fields) POST
shop.feature.delete ($feature_id) POST

tag

shop.tag.getList (*$storefront*)

set

shop.set.getList ( ) GET
shop.set.getInfo ($set_id) GET
shop.set.add (data fields) POST
shop.set.update ($set_id, data fields) POST
shop.set.delete ($set_id) POST

order

customer

coupon

cart

service

tax

stock
shop.stock.getList ( )
shop.stock.getInfo ($stock_id) GET

Stickies API

To illustrate use of the Webasyst API, several methods have also been implemented for the “Stickies” app: http://www.webasyst.com/apps/stickies/

stickies.sheet.getList( )
stickies.sheet.getInfo($id)
stickies.sheet.add(*data fields*)
stickies.sheet.update($id, *data fields*)
stickies.sheet.delete($id)

stickies.sticky.getList($sheet_id)
stickies.sticky.getInfo($id)
stickies.sticky.add(*data fields*)
stickies.sticky.update($id, *data fields*)
stickies.sticky.delete($id)