PHP class description for developers:
API connection and database sync.

Name:
SalesLayer_Updater (version: 1.20)

This connector expands the "SalesLayer-Conn.php" class.

This class contains all the logic and connection methods for Sales Layer's customizable APIs.

Important Atención:
Version 1.20 of SalesLayer_Updater contains updates meaning that table management may cause errors when working with previous versions. We recommend deleting the tables and regenerating them using the new version of SalesLayer_Updater.

Object creation:

$sl_api = **new SalesLayer_Updater** ($dbname, $dbusername, $dbpassword, $dbhost, $channel_code, $secret_key);

This object connects to the API and saves data in the local database.

$channel_code is the ID number of the Sales Layer connector which we are configuring.
$secret_key is the private key used to securely verify the transactions with the Sales Layer API. The parameters **$dbname**, **$dbusername**, **$dbpassword**, **$dbhost** are required in order to connect to the database.

Main API calls:

$sl_api->**update** ();

Connects to Sales Layer's API, checks if any modifications to the data have been made, and then updates the local database with the changes. This object also creates the tables that will store data.

$sl_api->**extract** ($table, $fields, $language, $conditions, $force_default_language, $order, $group, $limit*, $get_internal_ids, $get_internal_names, $get_channel_id*);

Runs data readings on the tables of the local database. This object is responsible for responding to the data in any required language. You can also build complex selection clauses for performing searches.

The parameter **$fields** is a matrix of the column names to be extracted. If this is left blank, or as null, then the function will return the information from every column.

In the parameter **$language** we can specify in which language we want to extract information, in the case of having our information stored in multiple languages.

The parameter **$conditions** is a matrix which can be used to filter the returned results, it's possible to add multiple conditions at once. There are three types of conditions: general search, searches by field, and grouped searches:

Examples:

```
$files          = [ 'section_name', 'section_description' ];
$conditions = [ [ 'field'  => 'color', 'search' => 'red' ], [ 'field'  => 'referece', 'strict' => true,  'condition' => '!=', 'value'
=>'RF45' ] ];
$result         = $SLupdate->extract('catalogue', $files, 'en', $conditions);
```

This condition searches for the word "red" in the columns: "name", "description", "color" and "size":

```
$conditions = [
    0 => [
        'search' => 'red',
        'field'     => 'name,description,color,size'
    ]
];
```

This condition returns all the items which have a value in the price field, equal or superior to 120

```
$conditions = [
    0 => [
        'value'       => 120,
        'condition' => '>=',
        'field'         => 'price'
    ]
];
```

This condition returns all the items which have the word "cáñamo" (exactly as it is written here) in the columns "material" or "furniture":

```
$conditions = [
    0 => [
        'value' => 'cáñamo',
        'strict'  => true,
        'field'   => 'material,furniture'
    ]
];
```

We can also use "group" to group conditions together:

```
$conditions = [
    0 => [  'value' => 120,  'condition' => '>=',  'field'  => 'price' ],
    1 => [ 'group' => 'and', ← or: or, not and xor  ],
    2 => [ 'search' => 'red',  'field' => 'name,description,color' ],
    3 => [  'search' => 'XL',  'field'     => 'size'  ],
    4 => [  'group' => 'close'  ]
];
```

In the Boolean parameter **$force_default_language** we can define whether we want to auto-fill any empty multi-language fields with existing values from completed, related fields in the default language.

In **$group** we can define whether we want or not to group various columns together.

In **$order** we can specify the order the item we want our items to be returned in. For example:

```
$order = [
    ‘name’ => ‘ASC’,
    ‘price’ => ‘DESC’
];
```

The parameter **$limit** is used to impose limits on the pagination of the results. For instance:

```
$limit = [
    ‘page’ => 1,
    ‘limit’  => 100
];
```

The boolean parameter **$get_internal_ids** can be used to specify whether or not we want the output data to include the internal IDs of the records. *By default this is set to “false”.*

The boolean parameter **$get_internal_names** can be used to define whether or not the column names used in the output data should be the exact names used for the fields in the database. *By default this is set to “false”.*

The boolean parameter **$get_channel_id** can be used to define if the result should include the ID of the connector from the database. This is useful when we have multiple connector configured in the same database. *By default this is set to “false”.*

Other API calls (version 1.29):

$sl_api->**database_connect** ($database, $username, $password, $hostname);

Connects to the local database.

$sl_api->**get_database_table_schema** ($table, $extended);

Returns the data schema of the stored information.

$sl_api->**get_database_table_joins** ($table);

Returns the tables linked to the specified table.

$sl_api->**has_database_table** ($table);

This call checks if a table returned by the API exists in the local database.

$sl_api->**create_database_table** ($table, $schema);

Creates a table in the local database based on the schema returned by the API.

$sl_api->**update_database_table** ($table, $schema);

Updates the table with a new modified schema.

$sl_api->**get_database_table_ids** ($table);

Returns a list with the IDs of each record stored in a table of the local database.

$sl_api->**get_default_language** ();

Returns the default language.

$sl_api->**get_languages** ();

Returns a list of the available languages.

$sl_api->**update_database_table_data** ($table, $data);

Updates a local database table with the new records.

$sl_api->**delete_all** ();

Deletes all the imported data and tables.

$sl_api->**get_database_calls** ();

Shows in a matrix all the calls to the database.

$sl_api->**delete_connector** ($code, $clean_items=false);

Removes a connector and all the associated items in the database. If you only want to delete the reference to the connector not the items, the second parameter ($clean_items) must be set as true.

$sl_api->**print_debug** ();

Prints the debugging information, if this debugging is enabled.

$sl_api->**get_field_titles** ();

Returns the titles of the field.

$sl_api→**get_language_field_titles** ();

Returns the titles of the fields in a specific language.

New calls (as of version 1.20):

$sl_api→**get_database_table_db_name** ($table);

Returns the name of the table in the database.

$sl_api→**get_table_fields_db** ($table);

Returns a list of the field names of a table in the database.

$sl_api→**get_table_fields_db_rels** ($table);

Returns an array with the field names from the database and the names of the corresponding field in the specified table.

$sl_api→**get_db_field_ID** ($table);

Returns the name of the field which is the primary key is the specified table.

$sl_api→**get_db_field_parent_ID** ($table);

Returns the name of the field with the value of the Category Parent ID as well as the main ID of the table. This field, Category Parent ID, is only found in the Categories table and is used to generate the category and subcategory structure..

$sl_api→**get_db_field_by_name** ($name, $table);

Returns, from the name of a field in a connector, its name in the database,

$sl_api→**get_default_language** ();

Returns the default language code of the connector.

$sl_api→**get_languages** ();

Returns a list of the language codes of a connector.

$sl_api→**get_table_title** ($language, $table);

Returns the title of the specified table in the specified language, if this has been defined in the connector, if not then it returns the default connector.

$sl_api→**get_language_titles_of_table** ($table);

Returns a list of all the titles of the specified table in each different language used.

$sl_api→**get_language_titles_of_fields** ($table = null);

Returns a list of all the fields in the spècified table with the names of all the fields in each different language used. If no table is specified then it returns all the names for every table.

$sl_api→**get_titles_of_fields** ($language, $table = null);

Returns the names of the fields in the specified language of the specified table, or of every table if no table is specified.

Other calls (inherited from **saleslayer_conn**) class:

Important:
Version 1.29 of SalesLayer_Calls contains updates which are incompatible with previous versions when it comes to returning lists of results.

$sl_api->**get_response_time** ($mode);

UNIX date returned. $mode (by default “datetime”) allows to indicate if you prefer date in UNIX format or “Y-m-d h:m:s” format. *Returns "false" if the request has not been done or occurs an error.*

$sl_api->**get_response_api_version** ();

Version of the API. *Returns "false" if the request has not been done or occurs an error.*

$sl_api->**get_response_action** ();

API action. *Returns "false" if the request has not been done or occurs an error.*

$sl_api->**has_response_error** ();

Checks if the API connection had an error. *Returns "**true**" if there was an error.*

$sl_api->**get_response_error** ();

Returns the error code generated by the API. *Returns "0" if there were no errors. You can find more information about error codes at the end of the document.*

$sl_api->**get_response_error_message** ();

Returns the error text. *If there was no error, returns an empty chain*.

$sl_api->**get_info** ($last_update, $params);

Sends a connection request to the API and prepares results. The variable $last_update (optional), indicates the UNIX timestamp of the last API request *(this timestamp can be obtained in every request, $last_update=$sl_api->**get_response_time**('unix'), and must be stored for future API requests: $sl_api->**get_info**($last_update))*.

$params allows you to send to the API any other extra parameters which are not data updates

Response: **true** if everything went well.

The untreated raw data can be read via: $sl_api→data_returned.

$sl_api->**get_response_field_titles** ();

Returns the titles of the field.

$sl_api→**get_response_language_field_titles** ();

Returns the titles of the fields in a specific language.

$sl_api->**get_response_table_information** ($table);

Returns a list of fields by table with the information of the field (string, numeric, boolean, datetime, image or file) and if the field is multi-language. The basic structure returned is the following:

```
[
  '{table}' =>[
    'fields' => [
      '{field name}' => [
        'type' => "{key|string|numeric|boolean|datetime|image|file|json}",
        ‘sanitized’ => “{sanitized name for database}”,
        'has_multilanguage' => {0|1},
        ‘title’ => “Public title”, ⟵ if has_multi-language = 1
        ‘titles’ => [ {field titles in the languages of the catalog} ], ⟵ if has_multi-language = 0
        'language_code' => '{en|es|fr|...}', ⟵ if has_multi-language = 1
        'basename' => '{name of the field without language code}', ⟵ if has_multi-language = 1
        'image_sizes' => [
          '{extension}' => [ 'width', 'height' ],
          ...
        ]
      ],
      ...
    ],
    'table_joins' => [

      'ID_{table}' => '{table}',
      ...
    ],
    'count_registers' => {total number of modified/deleted records},
    'count_modified' => {number of modified records},
    'count_deleted' => {number of deleted records}
  ],
  ...
]
```

$sl_api->**get_response_default_language** ();

Returns the default language code.

$sl_api->**get_response_languages_used** ();

Returns a matrix with the language codes used for multi-language fields.

$sl_api->**get_response_table_deleted_ids** ($table);

Returns a matrix with the deleted IDs of the table. *"**False**" if there are no deleted records.*

$sl_api->**get_response_table_modified_ids** ($table);

Returns a matrix with the modified IDs of the table. *"**False**" if there are no modified records.*

$sl_api->**get_response_table_modified_data** ($table);

Returns a matrix with just those records which have been modified. *"**False**" if there are no modified records.*

$sl_api->**get_response_list_modified_files** ();

Returns a matrix with the images or files to download. This option is designed to be used by apps that need to know how many files are about to be downloaded. *"**False**" if there are no modified files.*

$sl_api->**get_response_table_data** ($table);

Returns a matrix with the information inside any table. The matrix is structured as follows:

```
[
  'modified' => [
    '0' => [
      'id' => {ID record identifier},
      'id_parent' * => {parent ID section},
      'id_{table name}' ** => {ID of the linked table (join)}, ← special field for nested tables
      'data' => [
        {key 1} => {value 1},
        {key 2} => {value 2},
        ...
      ],
    ],
    '1' =>[ ... ],
    ...
  ],
  'deleted' => [
    {ID 1}, {ID 2}, {ID 3}, ...
  ]
];
```

** = The field 'id_parent' allows nesting hierarchically the categories tree of the catalogue, or in sales material tables.*

*** = The field 'ID_{table name}' only will be provided in the table 'product_formats' to link with 'products' and other nested sales material tables. The table 'product_formats' stores product variations for example: the different sizes, colours, fabrics... of a shirt.*

$sl_api->**get_response_offline_file** ();

Returns the TAR with all the files packed. This option is intended for apps that need an offline mode. *Returns "**false**" if the file does not exist.*

$sl_api->**get_response_connector_Schema** ();

Returns the configuration schema of the connector. May change on very specific configurations, but basically contains:

```
[
    'languages' => [ {codes of the languages used} ],
    'default_language' => {default language},
    'offline_mode' => boolean field that indicates if a TAR file with all the files packed exists (used for apps with
offline mode).
    'connector_type' => kind of connector (by default: 'default')
]
```

$sl_api->**get_response_waiting_files** ();

Returns the number of images or files being or waiting to be processed:

```
[
    'waiting' => [
        'images' => [
            'total' => {number of entire images in process},
            'partial' => {number of images without any size}
        ],
        'files' => {number of files in process}
        ]
    ]
];
```

$sl_api->**set_info** ($update_items*[, $delete_items, $compression]*);

Makes a connection request to the API and sends the data contained in **$update_items** so that they can be updated in Sales Layer. The structure contained in $update_items must be:

```
"{table name}" = [
        [
                "{reference field}" = "{unique item reference}"
                "{field name 1}" = "{new value}",
                "{field name 2}" = "new value}",
                ...
        ],
        … next item to update ...
],
"{next table name}" = [
        … list of items to update ...
]
```

It is also possible to attach the URL of a CSV file:

```
"{table name}" = "https://{CSV file URL with the data to be updated}",
"{next table name}" = "https://{CSV file URL with the data to be updated}"
```

Error codes:

These error codes are shown by $sl_api->**get_response_error**() y $sl_api->**get_response_error_message**().

The following table shows the defined errors:

1	Validation error	Error
2	Invalid connector code ('code')	Error
3	Invalid unique key ('unique')	Error
4	Invalid codification key ('key')	Error
5	Date of last update incorrect	Informative, returns all data
6	The specified API version does not exist	Informative
7	Invalid output mode ('output')	Informative, returns all data in JSON
8	Invalid compression type ('compression')	Informative, does not compress.
9	Invalid private key	Error
10	Service temporarily blocked	Alert
11	Service temporarily not available	Alert
12	Invalid timestamp ('time')	Error
13	Expired timestamp ('time')	Error
14	Updating data. Try later	Informative
101	Empty response, incomplete or incorrectly formatted	Error
102	Network connection error	Error
103	Local database connection error	Error
104	Local database access error	Error
105	The ID code of the connector does not correspond with the provided connector	Error