Getting Started

A few thoughts about the project and the documentation. Setting up your project.

welcome

Welcome to the IGDB Wrapper documentation! This documentation will cover all of the functionalities of this wrapper with lots of example codes. To personalise these codes you can add your own tokens to this documentation on the main page (click on the logo in the top left corner to go back). This way every code example where the tokens are required will contain your own tokens.

The wrapper's main purpose is to provide a simple solution to fetch data from IGDB's database using PHP. The wrapper contains endpoint methods for every IGDB API endpoints and even more!

To have access to IGDB's database you have to register a Twitch Account and have your own `client_id` and `access_token`. Refer to the Account Creation (https://api-docs.igdb.com/#account-creation) and Authentication (https://api-docs.igdb.com/#authentication) sections of the IGDB API Documentation (https://api-docs.igdb.com/) for details.

Using the documentatioycx

The documentation contains a lot of copy-paste example codes. In most cases, the code snippets contains the instantiation of the `IGDB` class. The constructor of this class will require you to pass your `client_id` and `access_token`. If you would like to have example codes with your personal tokens you can provide these on the front page of the documentation. To get back to the main page, just click the logo on the top left corner.

Click the link on the homescreen to set your tokens

Clicking the link will open the window where you can see additional information about this feature and two fields where you can store your tokens.

The window where the tokens can be stored

> If you chose to save the tokens these will be stored only in your browser locally! If the tokens are not saved the data will be erased when your browser is closed.

After your tokens are set the example codes will contain your tokens which are ready to copy.

The tokens are set in the example code

If you want to delete your tokens you can click the Delete the tokens button anytime.

Setting up your project

The project has multiple PHP files which are in the `src` folder of the repository:

- **IGDB.php**: the IGDB wrapper itself
- **IGDBEndpointException.php**: an exception thrown by the IGDB wrapper endpoint methods
- **IGDBInvalidParameterException.php**: an exception thrown by the Query Builder configuring methods
- **IGDBQueryBuilder.php**: the Query Builder
- **IGDBUtil.php**: utility and helper methods

These files can be imported one-by-one but it is highly recommended to import the `class.igdb.php` file which handles all the imports for you.

```
<?php

    require_once "class.igdb.php";

?>
```

After the import, you are ready to use the wrapper.

> :warning If you decide to import the files separately, make sure to check the source code since the exception classes are imported in the respective files where they are needed with hardcoded paths. The files has to be in the same folder to make it work!

Instantiating the wrapper

The wrapper will need your `client_id` and generated (https://api-docs.igdb.com/#authentication) `access_token`.

```
<?php

    require_once "class.igdb.php";

    $igdb = new IGDB("{client_id}", "{access_token}");

?>
```

> The wrapper itself does not validate your tokens. If your credentials are invalid you will get an `IGDBEndpointException` from the endpoint methods when you are sending your queries.

Building Queries

There is a helper class to build apicalypse queries called `IGDBQueryBuilder`. With this class you can easily build parameterized queries.

```php
<?php

    require_once "class.igdb.php";

    $builder = new IGDBQueryBuilder();

    try {
        $query = $builder
            ->search("uncharted 4")
            ->fields("id, name")
            ->limit(1)
            ->build();
    } catch (IGDBInvalidParameterException $e) {
        // Invalid argument passed to one of the methods
        echo $e->getMessage();
    }
?>
```

The parameters in the above example are valid, so the `$query` variable will hold the valid apicalypse string.

```
fields id,name; search "uncharted 4"; limit 1;
```

> For more details on the Query Builder refer to the IGDB Query Builder section of this documentation.

Sending Queries to IGDB

When your `$igdb` object is created you can send your queries to the IGDB database right away. There are multiple endpoint methods which will allow you to fetch data from the respective IGDB Endpoint (https://api-docs.igdb.com/#endpoints). These endpoint methods in the wrapper will accept apicalypse (https://api-docs.igdb.com/#apicalypse-1) formatted query strings.

```php
<?php

    require_once "class.igdb.php";

    $igdb = new IGDB("{client_id}", "{access_token}");

    try {
        $result = $igdb->game('search "uncharted 4"; fields id,name; limit
1;');
    } catch (IGDBEndpointException $e) {
        // Something went wrong, we have an error
        echo $e->getMessage();
    }

?>
```

Since the query is valid we have the `$result` array containing our matched records.

```
array (size=1)
  0 =>
    object(stdClass)[2]
      public 'id' => int 7331
      public 'name' => string 'Uncharted 4: A Thief's End' (length=26)
```

> For more details on the IGDB Wrapper refer to the IGDB Wrapper section of this documentation.

Handling errors

There are two main type of errors which could occur when using the wrapper:

- **"local" errors**: when using the Query Builder to build your queries you may pass invalid parameters to the configuring methods. These methods could throw `IGDBInvalidParameterException` if an invalid parameter or invalid type of parameter is passed. Head to the Handling Builder Errors section to find out more.
- **"remote" errors**: when a query is sent to the IGDB API, but for some reason the query fails. This happens only, when a non-successful response code is recieved from the API. In this case the endpoint methods will throw an `IGDBEndpointException`. Head to the Handling Request Errors section to find out more.

IGDB Query Builder

Everything about the Query Builder. Configuring and building your queries.

Instantiating the Builder

To instantiate the Builder simply use the `new` keyword.

```php
<?php

    require_once "class.igdb.php";

    $builer = new IGDBQueryBuiler();

?>
```

Now the `$builder` object will expose every configuring methods to set up your query before building it.

Configuring the Builder

There are two ways to configure the builder.

Builder approach

The Builder is using a builder pattern to configure itself with the configuring methods before parsing the properties to a query string. When every parameter is set, calling the `build()` method will start processing the parameters and will return the query string itself.

> :warning When using the builder approach, you must enclose your configuring methods in try...catch blocks. Read more on this in the Handling Builder Errors section.

```php
<?php

    require_once "class.igdb.php";

    $builder = new IGDBQueryBuilder();

    try {
        $query = $builder
            ->search("uncharted 4")
            ->fields("id, name")
            ->limit(1)
             // make sure to call the build method to parse the query string
            ->build();
    } catch (IGDBInvaliParameterException $e) {
        // invalid value passed to a method
        echo $e->getMessage();
    }

?>
```

Traditional approach

To support the traditional approach of configuring the queries - the `$options` array - there is a method called `options()`. Passing your `$options` array to this method will process your parameters and will set up the builder.

> :warning Using the Builder this way is not recommended as this functionality may will be removed in future versions. Use the builder approach instead.

```php
<?php

    require_once "class.igdb.php";

    $builder = new IGDBQueryBuilder();

    $options = array(
        "search" => "uncharted 4",
        "fields" => "id,name",
        "limit" => 1
    );

    try {
        // pass your $options to the options method
        // then build the query
        $query = $builder->options($options)->build();
    } catch (IGDBInvaliParameterException $e) {
        // invalid key or value found in the $options array
        echo $e->getMessage();
    }

?>
```

Handling Builder Errors

When an invalid parameter is passed to any of the configuring methods, an `IGDBInvaliParameterException` is thrown. To properly handle these errors, at least the configuring methods should be enclosed in a try...catch block. Calling the `getMessage()` method of the exception object will return the error message.

```
<?php

    $builder = new IGDBQueryBuilder();

    try {
        $query = $builder
            ->search("uncharted")
            // passing an integer to the fields parameter is invalid
            // as this methods expects a string or an array of strings
            ->fields(1)
            ->build();

        echo $query;
    } catch (IGDBInvalidParameterException $e) {
        // catching the exception, printing the error message
        echo $e->getMessage();
    }

?>
```

The script above will produce this output:

```
Invalid type of parameter for fields! String or array of strings are expected,
integer passed!
```

It is important to keep in mind, that this validation is happening "locally" on your server. The query builder can only validate the type of variables you pass, or it can validate against a list of predefined values that the API expects. For example, if the example above is altered like this:

```
<?php

    $builder = new IGDBQueryBuilder();

    try {
        $query = $builder
            ->search("uncharted")
            // passing a field name that doesnt exist on the game endpoint
            ->fields("nonexistingfield")
            ->build();

        // no exception thrown, printing the query string
        echo $query;
    } catch (IGDBInvalidParameterException $e) {
        echo $e->getMessage();
    }

?>
```

The builder will not throw an exception as the method got a string, which is valid. The output of the script will be:

```
fields nonexistingfield; search "uncharted";
```

The field `nonexistingfield` does not exist on the `game` endpoint which will cause an error on the IGDB API. These errors has to be handled separately. Read more on this in the Handling Request Errors section.

Configuring Methods

The Builder class has its own configuring methods to set the parameters before building the query string. If any of these methods recieves an invalid argument they will throw an `IGDBInvalidArgumentException`. Make sure to set these parameters in a try...catch block.

Count

```
public function count(boolean $count = true) throws
IGDBInvalidArgumentException: IGDBQueryBuilder
```

This method will accept a boolean value. This parameter is processed **in case of a multiquery build only**. If this value is `true` the response from IGDB will contain the number of records matching the filters. If `false` the records will be returned.

> If the method is called without parameters, the value will be set to `true`.

Default value: by default this value is `false`, the queries will return the records instead of their count.

Parameters:

- `$count`: `true` if a record count is required, `false` otherwise.

Returns: `IGDBQueryBuilder` instance

```php
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // Configuring the query
        $builder
            ->name("Number of games")
            ->endpoint("game")
            // the default value of count is false
            // to retrieve the count of the matched records, set it to true
            // ->count(true) has the same result
            ->count();

            // if you wish to remove the count parameter for any reason later
on
            // you can call the count method with a false parameter
            // ->count(false)

        // building the query
        $query = $builder->build(true);
    } catch (IGDBInvaliParameterException $e) {
        // invalid key or value found in the $options array
        echo $e->getMessage();
    }

?>
```

The value of `$query`:

```
query games/count "Number of games" {
  fields *;
};
```

Custom Where

```php
public function custom_where(string $custom_where): IGDBQueryBuilder
```

This method will add a where statement to your query, but will not validate your input or throw any exceptions.

Parameters:

- `$custom_where`: an apicalypse string with a custom where statement

Returns: `IGDBQueryBuilder` instance

```php
<?php

    // traditional approach
    $options = array(
        "custom_where" => "(platforms = [6,48] & genres = 13) | (platforms = [130,48] & genres = 12)"
    );

    // builder approach
    $builder->custom_where("(platforms = [6,48] & genres = 13) | (platforms = [130,48] & genres = 12)");

?>
```

Endpoint

```php
public function endpoint(string $endpoint) throws IGDBInvalidArgumentException: IGDBQueryBuilder
```

This method will set the endpoint to send your **multiquery** against. This parameter is processed **in case of a multiquery build only**. When setting this value make sure to use the name of the endpoint instead of it's request path. (for example `game` instead of `games` and so on).

> :warning In case of a multiquery build this parameter is mandatory! If this is missing from the configuration the build will throw an `IGDBInvalidParameterException`! Refer to the build method for details.

Parameters:

- `$endpoint`: the name of the endpoint to send the query against. Make sure to use the name of the endpoint instead of it's request path! The list of endpoints for this parameter:
    - age_rating
    - alternative_name
    - artwork
    - character_mug_shot
    - character
    - collection
    - company_logo
    - company_website
    - company
    - cover
    - external_game
    - franchise
    - game_engine_logo
    - game_engine
    - game_mode
    - game_version_feature_value
    - game_version_feature
    - game_version
    - game_video
    - game
    - genre
    - involved_company

- keyword
- multiplayer_mode
- multiquery
- platform_family
- platform_logo
- platform_version_company
- platform_version_release_date
- platform_version
- platform_website
- platform
- player_perspective
- release_date
- screenshot
- search
- theme
- website

Returns: `IGDBQueryBuilder` instance

```
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // configuring the query
        $builder
            ->name("Game with ID of 25076")
            ->endpoint("game")
            ->fields("id,name")
            ->where("id = 25076");

        // building the query
        $query = $builder->build(true);
    } catch (IGDBInvaliParameterException $e) {
        // invalid key or value found in the $options array
        echo $e->getMessage();
    }

?>
```

The value of `$query`:

```
query games "Game with ID 25076" {
  fields id,name;
  where id = 25076;
};
```

Exclude

```
public function exclude(string | array $exclude) throws
IGDBInvalidArgumentException: IGDBQueryBuilder
```

Exclude is a complement to the regular fields which allows you to request all fields with the exception of any numbers of fields specified with exclude.

Parameters:

- `$exclude`: the list of the fields to exclude. This can be either a comma separated list of field names or an array of field names.

Returns: `IGDBQueryBuilder` instance

```
<?php

    // traditional approach, as a string
    $options = array(
        "exclude" => "name,slug"
    );

    // traditional approach, as an array
    $options = array(
        "exclude" => array("name", "slug")
    );

    // builder approach, as a string
    $builder->exclude("name,slug");

    // builder approach, as an array
    $builder->exclude(array("name", "slug"));

?>
```

Fields

```
public function fields(string | array $fields) throws
IGDBInvalidArgumentException: IGDBQueryBuilder
```

You can tell which fields you want to get in the result with this parameter.

Fields are properties of an entity. For example, a Game field would be genres or release_dates. Some fields have properties of their own, for example, the genres field has the property name.

Default value: by default every field (`*`) will be requested from the endpoint.

Parameters:

- `$fields`: the list of the required fields. This can be either a comma separated list of field names, or an array of field names.

Returns: `IGDBQueryBuilder` instance

```php
<?php

    // traditional approach, as a string
    $options = array(
        "fields" => "id,name"
    );

    // traditional approach, as an array
    $options = array(
        "fields" => array("id", "name")
    );

    // builder approach, as a string
    $builder->fields("id,name");

    // builder approach, as an array
    $builder->fields(array("id", "name"));

?>
```

ID

```php
public function id(array | int $id) throws IGDBInvalidArgumentException: IGDBQueryBuilder
```

This is kind of a traditional thing. Since the IGDB team introduced the apicalypse query there is no separate ID field. This method simply adds a where statement to your query.

```php
<?php

    $builder->id(1);
    // where id = 1;

    $builder->id(array(1,2,3));
    // where id = (1,2,3);

?>
```

Parameters:

- `$id`: one (integer) or more (array of integers) record ID's.

Returns: `IGDBQueryBuilder` instance

```php
<?php

    // traditional approach, single id
    $options = array(
        "id" => 1
    );

    // traditional approach, multiple id's
    $options = array(
        "id" => array(1,2,3)
    );

    // builder approach, single id
    $builder->id(1);

    // builder approach, multiple id's
    $builder->id(array(1,2,3));

?>
```

Limit

```php
public function limit(int $limit) throws IGDBInvalidArgumentException:
IGDBQueryBuilder
```

The maximum number of results in a single query. This value must be a number between `1` and `500`.

Default value: by default the IGDB API will return `10` results.

Parameters:

- `$limit`: a number between 1 and 500

Returns: `IGDBQueryBuilder` instance

```php
<?php

    // traditional approach
    $options => array(
        "limit" => 20
    );

    // builder approach
    $builder->limit(20);

?>
```

Name

```php
public function name(string $name) throws IGDBInvalidArgumentException:
IGDBQueryBuilder
```

This method will set a name for the query. This parameter is processed **in case of a multiquery build only**.

:warning In case of a multiquery build this parameter is **mandatory**! If this is missing from the configuration the build will throw an `IGDBInvalidParameterException`! Refer to the build method for details.

Parameters:

- `$name`: the name of the query

Returns: `IGDBQueryBuilder` instance

```
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // configuring the query
        $builder
            ->name("Game with ID of 25076")
            ->endpoint("game")
            ->fields("id,name")
            ->where("id = 25076");

        // building the query
        $query = $builder->build(true);
    } catch (IGDBInvaliParameterException $e) {
        // invalid key or value found in the $options array
        echo $e->getMessage();
    }

?>
```

The value of `$query`:

```
query games "Game with ID 25076" {
  fields id,name;
  where id = 25076;
};
```

Offset

```
public function offset(int $offset) throws IGDBInvalidArgumentException: IGDBQueryBuilder
```

This will start the result list at the provided value and will give `limit` number of results. This value must be `0` or greater.

Default value: the IGDB API will use `0` as a default offset value.

Parameters:

- `$offset`: a number which can be `0` or greater.

Returns: `IGDBQueryBuilder` instance

```
<?php

    // traditional approach
    $options = array(
        "offset" => 5
    );

    // builder approach
    $builder->offset(5);

?>
```

Options

```
public function options(array $options) throws IGDBInvalidArgumentException:
IGDBQueryBuilder
```

> :warning Using the Builder this way is not recommended as this functionality may be removed in future versions. Use the builder approach instead.

With this method you can parse your `$options` array instead of using the builder approach. If a non-valid key or value is in the array, an `IGDBInvalidParameterException` will be thrown.

Passing your `$options` array to this method will configure the builder with the parameters in it.

Parameters:

- `$options`: the options array with your configuration in it

Returns: `IGDBQueryBuilder` instance

```php
<?php

    require_once "class.igdb.php";

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    // setting up the options array
    $options = array(
        "search" => "uncharted 4",
        "fields" => "id,name",
        "limit" => 1
    );

    try {
        // configuring the builder with an options array
        $builder->options($options);

        // still have to call the build method to building the query
        $query = $builder->build();
    } catch (IGDBInvaliParameterException $e) {
        // invalid key or value found in the $options array
        echo $e->getMessage();
    }

?>
```

:warning Calling this method **will not reset** the configuration. Stacking the `options()` calls will add each parameter to the builder, overwriting the old values with the new ones. To clear the previous configuration items use the reset method.

```php
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // stacking the options calls will add each parameter as this not
resets the configuration
        $builder
            ->options(array("search" => "uncharted"))
            ->options(array("fields" => "id,name"))
            // this call will overwrite the search parameter!
            ->options(array("search" => "overwritten uncharted"))
            ->options(array("limit" => 1));

        // building the query
        $query = $builder->build();
    } catch (IGDBInvaliParameterException $e) {
        // invalid key or value found in the $options array
        echo $e->getMessage();
    }

?>
```

The query:

```
fields id,name; search "overwritten uncharted"; limit 1;
```

Reset

```
public function reset(): IGDBQueryBuilder
```

This method will reset every configuration to the default values.

Parameters: -

Returns: `IGDBQueryBuilder` instance

```php
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    // setting up the options1 array
    $options1 = array(
        "search" => "uncharted 4",
        "fields" => "id,name",
        "limit" => 2
    );

    // setting upt he options2 array
    $options2 = array(
        "search" => "star wars",
        "fields" => "cover,slug",
        "limit" => 1
    );

    try {
        // configuring the builder with options1, reset the values then pass
options2
        $builder->options($options1)->reset()->options($options2);

        // building the query
        $query = $builder->build();
    } catch (IGDBInvaliParameterException $e) {
        // invalid key or value found in the $options array
        echo $e->getMessage();
    }

?>
```

The code above will return the `$options2` array's configuration, as it got reset with the `reset()` method.

```
fields cover,slug; search "star wars"; limit 1;
```

Search

```
public function search(string $search) throws IGDBInvalidArgumentException:
IGDBQueryBuilder
```

Search based on name, results are sorted by similarity to the given search string.

Parameters:

- `$search`: the search string

Returns: `IGDBQueryBuilder` instance

```php
<?php

    // traditional approach
    $options = array(
        "search" => "uncharted 4"
    );

    // builder approach
    $builder->search("uncharted 4");

?>
```

Sort

```php
public function sort(string | array $sort) throws IGDBInvalidArgumentException: IGDBQueryBuilder
```

Sorting (ordering) is used to order results by a specific field. The parameter can be either an apicalypse formatted sort string or an array with specific key-value pairs. If you provide the Order parameter as an array, you must have two values in it with the following keys:

- `field`: The field you want to do the ordering by
- `direction`: The direction of the ordering. It must be either `asc` for ascending or `desc` for descending ordering.

Parameters:

- `$sort`: either an apicalypse sort string or an array with specific key-value pairs.

Returns: `IGDBQueryBuilder` instance

```php
<?php

    // traditional approach as a string
    $options = array(
        "sort" => "release_dates.date desc",
    );

    // traditional approach as an array
    $options = array(
        "sort" => array(
            "field" => "release_dates.date",
            "direction" => "desc"
        )
    );

    // builder approach as a string
    $builder->sort("release_dates.date desc");

    // builder approach as an array
    $builder->sort(
        array(
            "field" => "release_dates.date",
            "direction" => "desc"
        )
    );

?>
```

Where

```php
public function where(string | array $where) throws
IGDBInvalidArgumentException: IGDBQueryBuilder
```

Filters are used to swift through results to get what you want. You can exclude and include results based on their properties. For example you could remove all Games where the rating was below 80 `(where rating >= 80)`.

The where parameter can be either an apicalypse formatted string or an array with specific key-value pairs. If you provide the where parameters as an array, you must have three fix keys in it:

- `field`: The name of the field you want to apply the filter to.
- `postfix`: The postfix you want to use with the filter. Refer to the Available Postfixes (https://api-docs.igdb.com/#filters) section of the IGDB Filters Documentation for available postfixes.
- `value`: The value of the filter.

The where filters will be concatenated with **AND** operators (`&`).

> :success Multiple filter parameters can be applied to the same query. Check the examples below.

Parameters:

- `$where`: either an apicalypse formatted string or an array with specific keys

Returns: `IGDBQueryBuilder` instance

```php
<?php
```

```php
// traditional approach, single criteria as a string
$options = array(
    "where" => "release_dates.platform = 8"
);

// traditional approach, single criteria as an array
$options = array(
    "where" => array(
        "field" => "release_dates.platform",
        "postfix" => "=",
        "value" => 8
    )
);

// traditional approach, multiple criteria as a string
$options = array(
    "where" => array(
        "release_dates.platform = 8",
        "total_rating >= 70"
    );
);

// traditional approach, multiple criteria as an array
$options = array(
    "where" => array(
        array(
            "field" => "release_dates.platform",
            "postfix" => "=",
            "value" => 8
        ),
        array(
            "field" => "total_rating",
            "postfix" => ">=",
            "value" => 70
        )
    )
);

// builder approach, single criteria as a string
$builder->where("release_dates.platform = 8");

// builder approach, single criteria as an array
$builder->where(
    array(
        "field" => "release_dates.platform",
        "postfix" => "=",
        "value" => 8
    )
);

// builder approach, multiple criteria as a string
$builder
    ->where("release_dates.platform = 8")
```

```
        ->where("total_rating >= 70");

    // builder approach, multiple criteria as an array
    $builder
        ->where(
            array(
                "field" => "release_dates.platform",
                "postfix" => "=",
                "value" => 8
            )
        )
        ->where(
            array(
                "field" => "total_rating",
                "postfix" => ">=",
                "value" => 70
            )
        );
?>
```

> This method is trying to validate your input against some rules and will throw an `IGDBInvalidArgumentException` in case of any issues. If you need more flexibility or custom where statements with complex conditions in you queries use the Custom Where method instead.

Building the Query

```
public function build(boolean $multiquery = false) throws
IGDBInvalidParameterException: mixed
```

When the Builder object is configured properly the final step is to build the query to an apicalypse query string. The syntax of the query depends on which endpoint the query is executed against. The multiquery endpoint requires a few additional information and a slightly different syntax.

A `game` endpoint query:

```
fields id,name;
where id = 25076;
```

A `multiquery` to the game `game` endpoint:

```
query games "Main Game" {
  fields id,name;
  where id = 25076;
};
```

The build method accepts one `boolean` parameter. Using this parameter the build method decides which syntax to return. If a multiquery is required, a few extra fields has to be set for the builder:

- `name`: the name of the query choosen by the user
- `endpoint`: the endpoint to send the query against
- `count` [optional]: whether the records or their count is required. By default the value of this parameter is `false`.

> :warning In case of a multiquery request the properties name and endpoint are **mandatory**! If any of these are missing from the configuration, an `IGDBInvalidParameterException` will be thrown.

Parameters:

- `$multiquery`: if `true`, a multiquery will be returned, an endpoint query otherwise. The default value of this parameter is `false`.

> :warning If a non-boolean parameter is passed to the build method, an `IGDBInavlidParameterException` is thrown!

Returns: the apicalypse query string

An endpoint query example:

```
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // configuring the query
        $query = $builder
            ->search("uncharted 4")
            ->fields("id, name")
            ->limit(1)
            ->build();
    } catch (IGDBInvaliParameterException $e) {
        // invalid value passed to a method
        echo $e->getMessage();
    }

?>
```

The value of `$query`:

```
fields id,name;
search "uncharted 4";
limit 1;
```

A multiquery example:

```php
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // configuring the query
        $builder
            ->name("Game with ID of 25076")
            ->endpoint("game")
            ->fields("id,name")
            ->where("id = 25076")
            // note the true parameter
            ->build(true);
    } catch (IGDBInvaliParameterException $e) {
        // invalid key or value found in the $options array
        echo $e->getMessage();
    }

?>
```

The value of `$query`:

```
query games "Game with ID of 25076" {
  fields id,name;
  where id = 25076;
};
```

IGDB Wrapper

The main wrapper. Methods, endpoints, properties, configurations. Sending your queries to the IGDB API.

Instantiating the wrapper

After importing the dependencies you can instantiate the class with the `new` keyword by passing your tokens to the constructor.

```php
<?php

    require_once "class.igdb.php";

    $igdb = new IGDB("{client_id}", "{access_token}");

?>
```

> :info The wrapper itself does not validate your tokens. If your credentials are invalid, you will get an error from the IGDB API after executing a query.

Public Methods

These methods are exposed from the `IGDB` object.

Get Request Info

```
public function get_request_info()
```

After a query is executed, the latest request information will be stored and can be retrieved using this method.

> The new version of the IGDB API (v4) will return a http response code `429` when you exceed the limit of requests on the database (4 requests per second at the time of writing this documentation).

```
<?php

    require_once "class.igdb.php";

    $igdb = new IGDB("{client_id}", "{access_token}");

    try {
        $igdb->game('fields id,name; search "uncharted 4"; limit 1;');

        // getting the details of the latest query
        $request_info = $igdb->get_request_info();

        // showing the details
        var_dump($request_info);
    } catch (IGDBEndpointException $e) {
        echo $e->getMessage();
    }

?>
```

Details of the query can be fetched from the output of the `get_request_info()` method (for example, the HTTP Response code from `http_code`).

```
array (size=37)
  'url' => string 'https://api.igdb.com/v4/games' (length=29)
  'content_type' => string 'application/json;charset=utf-8' (length=30)
  'http_code' => int 200
  'header_size' => int 870
  'request_size' => int 224
  'filetime' => int -1
  'ssl_verify_result' => int 20
  'redirect_count' => int 0
  'total_time' => float 0.748895
  'namelookup_time' => float 0.000705
  'connect_time' => float 0.048049
  'pretransfer_time' => float 0.088656
  'size_upload' => float 46
  'size_download' => float 73
  'speed_download' => float 97
  'speed_upload' => float 61
  'download_content_length' => float 73
  'upload_content_length' => float 46
  'starttransfer_time' => float 0.088661
  'redirect_time' => float 0
  'redirect_url' => string '' (length=0)
  'primary_ip' => string '104.22.65.239' (length=13)
  'certinfo' =>
    array (size=0)
      empty
  'primary_port' => int 443
  'local_ip' => string '192.168.1.99' (length=12)
  'local_port' => int 58813
  'http_version' => int 3
  'protocol' => int 2
  'ssl_verifyresult' => int 0
  'scheme' => string 'HTTPS' (length=5)
  'appconnect_time_us' => int 88535
  'connect_time_us' => int 48049
  'namelookup_time_us' => int 705
  'pretransfer_time_us' => int 88656
  'redirect_time_us' => int 0
  'starttransfer_time_us' => int 88661
  'total_time_us' => int 748895
```

Construct URL

```
public function construct_url(string $endpoint, boolean $count = false) throws IGDBInvalidParameterException: string
```

The method will construct the full URL for the request and will return the constructed URL as a string. If an invalid endpoint name passed to `$endpoint` an `IGDBInvalidParameterException` will be thrown.

Parameters:

- `$endpoint`: the endpoint to use (the name of the endpoint, not the path to it!). Possible values:
    - age_rating
    - age_rating_content_description

    - alternative_name
    - artwork
    - character
    - character_mug_shot
    - collection
    - collection_membership
    - collection_membership_type
    - collection_relation
    - collection_relation_type
    - collection_type
    - company
    - company_logo
    - company_website
    - cover
    - event
    - event_logo
    - event_network
    - external_game
    - franchise
    - game
    - game_engine
    - game_engine_logo
    - game_localization
    - game_mode
    - game_version
    - game_version_feature
    - game_version_feature_value
    - game_video
    - genre
    - involved_company
    - keyword
    - language
    - language_support
    - multiplayer_mode
    - platform
    - language_support_type
    - platform_family
    - network_type
    - platform_logo
    - platform_version_company
    - platform_version
    - platform_website
    - platform_version_release_date
    - player_perspective
    - region
    - release_date
    - release_date_status
    - screenshot
    - search
    - theme
    - website
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: the full constructed URL to the IGDB Endpoint as a string.

```php
<?php

    // instantiating the wrapper
    $igdb = new IGDB("{client_id}", "{access_token}");

    try {
        // constructing an url
        $url = $igdb->construct_url("game");

        // constructing an url to get counts
        $count_url = $igdb->construct_url("game", true);

        // showing the results
        echo "url: " . $url;
        echo "count url: " . $count_url;
    } catch (IGDBInvalidParameterException $e) {
        // an invalid parameter passed to the construct_url method
        echo $e->getMessage();
    }

?>
```

Output:

```
url: https://api.igdb.com/v4/games
count url: https://api.igdb.com/v4/games/count
```

Close CURL Session

```php
public function curl_close(): void
```

You can close the CURL session handler manually if you need to.

Parameters: -

Returns: -

> The curl handler will be reinitialized automatically when a new request is sent to the IGDB API with any of the endpoint methods.

Reinitialize CURL Session

```php
public function curl_reinit() : void
```

After you closed the CURL session manually with curl_close() you can manually reinitialize the curl handler.

Parameters: -

Returns: -

> Before sending a request with an endpoint method, the wrapper will check the status of the curl handler. If it is closed, it will reinitialize it automatically.

Handling Request Errors

Your query may fail on the IGDB side. In this case the API will send back a non-successful response code indicating that something went wrong. When this happens an `IGDBEndpointException` can be caught to extract information about the issue. To catch these errors you have to enclose your endpoint method calls in a try...catch block.

```
<?php

    $igdb = new IGDB("{client_id}", "{access_token}");

    // your query string with a field that doesn't exist
    $query = 'search "uncharted"; fields nonexistingfield;';

    try {
        // executing the query
        $games = $igdb->game($query);
    } catch (IGDBEndpointException $e) {
        // since the query contains a non-existing field, an error occured
        // printing the response code and the error message
        echo "Response code: " . $e->getResponseCode();
        echo "Message: " . $e->getMessage();
    }

?>
```

Since the query above is not valid, as there is no field called `nonexistingfield` on the game endpoint, the API will send a response with an error message and a non-successful response code. The result of the script above is:

```
Response code: 400
Message: Invalid Field
```

You can also get some additional information about this request using the `get_request_info()` method.

Endpoints

Every endpoint method is named after the IGDB API endpoints using snake-casing naming convention. These methods are expecting at least one parameter, the `$query` itself. The second `$count` parameter is optional, it is `false` by default.

Parameters:

- `$query`: the query itself as an apicalypse string
- `$count`: a `boolean` value to whether return the records or the count of the records

:success To build your queries, give IGDB Query Builder a try!

These methods will return **an array of objects** decoded from IGDB response JSON when the `$count` parameter is false. Otherwise, it will execute a count query against the selected endpoint which will return an object with a `count` property holding the sum of the found items. The count queries can be filtered with where parameters.

`IGDBEndpointException` is thrown if a non-successful response code is recieved from the IGDB API. To find out how to handle request errors, head to the Handle Request Errors section.

Please refer to the return values section for more details about the return values of these methods.

For the endpoint specific fields that the API returns please refer to the IGDB documentation's respective paragraph. Each endpoint has a direct link!

Age Rating

```
public function age_rating(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Age Rating (https://api-docs.igdb.com/#age-rating) endpoint.

Endpoint Description: Age Rating according to various rating organisations

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Age Rating endpoint method
    $igdb->age_rating($query, $count);

?>
```

Age Rating Content Description

```
public function age_rating_content_description(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Age Rating Content Description (https://api-docs.igdb.com/#age-rating-content-description) endpoint.

Endpoint Description: Age Rating Descriptors

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Age Rating Content Description endpoint method
    $igdb->age_rating_content_description($query, $count);

?>
```

Alternative Name

```
public function alternative_name(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Alternative Name (https://api-docs.igdb.com/#alternative-name) endpoint.

Endpoint Description: Alternative and international game titles

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Alternative Name endpoint method
    $igdb->alternative_name($query, $count);

?>
```

Artwork

```
public function artwork(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Artwork (https://api-docs.igdb.com/#artwork) endpoint.

Endpoint Description: official artworks (resolution and aspect ratio may vary)

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Artwork endpoint method
    $igdb->artwork($query, $count);

?>
```

Character

```
public function character(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Character (https://api-docs.igdb.com/#character) endpoint.

Endpoint Description: Video game characters

Parameters:

- `$query`: an apicalypse formatted query String

- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Character endpoint method
    $igdb->character($query, $count);

?>
```

Character Mug Shot

```
public function character_mug_shot(string $query, boolean $count = false)
throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Character Mug Shot (https://api-docs.igdb.com/#character-mug-shot) endpoint.

Endpoint Description: Images depicting game characters

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Character Mug Shot endpoint method
    $igdb->character_mug_shot($query, $count);

?>
```

Collection

```
public function collection(string $query, boolean $count = false) throws
IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Collection (https://api-docs.igdb.com/#collection) endpoint.

Endpoint Description: Collection, AKA Series

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Collection endpoint method
    $igdb->collection($query, $count);

?>
```

Collection Membership

```php
public function collection_membership(string $query, boolean $count = false)
throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Collection Membership (https://api-docs.igdb.com/#collection-membership) endpoint.

Endpoint Description: The Collection Memberships.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Collection Membership endpoint method
    $igdb->collection_membership($query, $count);

?>
```

Collection Membership Type

```php
public function collection_membership_type(string $query, boolean $count =
false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Collection Membership Type (https://api-docs.igdb.com/#collection-membership-type) endpoint.

Endpoint Description: Enums for collection membership types.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Collection Membership Type endpoint method
    $igdb->collection_membership_type($query, $count);

?>
```

Collection Relation

```
public function collection_relation(string $query, boolean $count = false)
throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Collection Relation (https://api-docs.igdb.com/#collection-relation) endpoint.

Endpoint Description: Describes Relationship between Collections.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Collection Relation endpoint method
    $igdb->collection_relation($query, $count);

?>
```

Collection Relation Type

```
public function collection_relation_type(string $query, boolean $count = false)
throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Collection Relation Type (https://api-docs.igdb.com/#collection-relation-type) endpoint.

Endpoint Description: Collection Relation Types

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Collection Relation Type endpoint method
    $igdb->collection_relation_type($query, $count);

?>
```

Collection Type

```
public function collection_type(string $query, boolean $count = false) throws
IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Collection Type (https://api-docs.igdb.com/#collection-type) endpoint.

Endpoint Description: Enums for collection types.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Collection Type endpoint method
    $igdb->collection_type($query, $count);

?>
```

Company

```
public function company(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Company (https://api-docs.igdb.com/#company) endpoint.

Endpoint Description: Video game companies. Both publishers & developers

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Company endpoint method
    $igdb->company($query, $count);

?>
```

Company Logo

```
public function company_logo(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Company Logo (https://api-docs.igdb.com/#company-logo) endpoint.

Endpoint Description: The logos of developers and publishers

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Company Logo endpoint method
    $igdb->company_logo($query, $count);

?>
```

Company Website

```php
public function company_website(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Company Website (https://api-docs.igdb.com/#company-website) endpoint.

Endpoint Description: Company Website

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Company Website endpoint method
    $igdb->company_website($query, $count);

?>
```

Cover

```php
public function cover(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Cover (https://api-docs.igdb.com/#cover) endpoint.

Endpoint Description: The cover art of games

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Cover endpoint method
    $igdb->cover($query, $count);

?>
```

Event

```
public function event(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Event (https://api-docs.igdb.com/#event) endpoint.

Endpoint Description: Gaming event like GamesCom, Tokyo Game Show, PAX or GSL

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Event endpoint method
    $igdb->event($query, $count);

?>
```

Event Logo

```
public function event_logo(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Event Logo (https://api-docs.igdb.com/#event-logo) endpoint.

Endpoint Description: Logo for the event

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Event Logo endpoint method
    $igdb->event_logo($query, $count);

?>
```

Event Network

```
public function event_network(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Event Network (https://api-docs.igdb.com/#event-network) endpoint.

Endpoint Description: Urls related to the event like twitter, facebook and youtube

Parameters:

- `$query`: an apicalypse formatted query String

- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Event Network endpoint method
    $igdb->event_network($query, $count);

?>
```

External Game

```
public function external_game(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the External Game (https://api-docs.igdb.com/#external-game) endpoint.

Endpoint Description: Game IDs on other services

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // External Game endpoint method
    $igdb->external_game($query, $count);

?>
```

Franchise

```
public function franchise(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Franchise (https://api-docs.igdb.com/#franchise) endpoint.

Endpoint Description: A list of video game franchises such as Star Wars.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Franchise endpoint method
    $igdb->franchise($query, $count);

?>
```

Game

```php
public function game(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game (https://api-docs.igdb.com/#game) endpoint.

Endpoint Description: Video Games!

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Game endpoint method
    $igdb->game($query, $count);

?>
```

Game Engine

```php
public function game_engine(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game Engine (https://api-docs.igdb.com/#game-engine) endpoint.

Endpoint Description: Video game engines such as unreal engine.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Game Engine endpoint method
    $igdb->game_engine($query, $count);

?>
```

Game Engine Logo

```
public function game_engine_logo(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game Engine Logo (https://api-docs.igdb.com/#game-engine-logo) endpoint.

Endpoint Description: The logos of game engines

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Game Engine Logo endpoint method
    $igdb->game_engine_logo($query, $count);

?>
```

Game Localization

```
public function game_localization(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game Localization (https://api-docs.igdb.com/#game-localization) endpoint.

Endpoint Description: Game localization for a game

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Game Localization endpoint method
    $igdb->game_localization($query, $count);

?>
```

Game Mode

```
public function game_mode(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game Mode (https://api-docs.igdb.com/#game-mode) endpoint.

Endpoint Description: Single player, Multiplayer etc

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Game Mode endpoint method
    $igdb->game_mode($query, $count);

?>
```

Game Version

```
public function game_version(string $query, boolean $count = false) throws
IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game Version (https://api-docs.igdb.com/#game-version) endpoint.

Endpoint Description: Details about game editions and versions.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Game Version endpoint method
    $igdb->game_version($query, $count);

?>
```

Game Version Feature

```
public function game_version_feature(string $query, boolean $count = false)
throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game Version Feature (https://api-docs.igdb.com/#game-version-feature) endpoint.

Endpoint Description: Features and descriptions of what makes each version/edition different from the main game

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Game Version Feature endpoint method
    $igdb->game_version_feature($query, $count);

?>
```

Game Version Feature Value

```php
public function game_version_feature_value(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game Version Feature Value (https://api-docs.igdb.com/#game-version-feature-value) endpoint.

Endpoint Description: The bool/text value of the feature

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Game Version Feature Value endpoint method
    $igdb->game_version_feature_value($query, $count);

?>
```

Game Video

```php
public function game_video(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Game Video (https://api-docs.igdb.com/#game-video) endpoint.

Endpoint Description: A video associated with a game

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Game Video endpoint method
    $igdb->game_video($query, $count);

?>
```

Genre

```
public function genre(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Genre (https://api-docs.igdb.com/#genre) endpoint.

Endpoint Description: Genres of video game

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Genre endpoint method
    $igdb->genre($query, $count);

?>
```

Involved Company

```
public function involved_company(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Involved Company (https://api-docs.igdb.com/#involved-company) endpoint.

Endpoint Description:

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Involved Company endpoint method
    $igdb->involved_company($query, $count);

?>
```

Keyword

```
public function keyword(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Keyword (https://api-docs.igdb.com/#keyword) endpoint.

Endpoint Description: Keywords are words or phrases that get tagged to a game such as "world war 2" or "steampunk".

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Keyword endpoint method
    $igdb->keyword($query, $count);

?>
```

Language

```
public function language(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Language (https://api-docs.igdb.com/#language) endpoint.

Endpoint Description: Languages that are used in the Language Support endpoint.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Language endpoint method
    $igdb->language($query, $count);

?>
```

Language Support

```
public function language_support(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Language Support (https://api-docs.igdb.com/#language-support) endpoint.

Endpoint Description: Games can be played with different languages for voice acting, subtitles, or the interface language.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Language Support endpoint method
    $igdb->language_support($query, $count);

?>
```

Multiplayer Mode

```php
public function multiplayer_mode(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Multiplayer Mode (https://api-docs.igdb.com/#multiplayer-mode) endpoint.

Endpoint Description: Data about the supported multiplayer types

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Multiplayer Mode endpoint method
    $igdb->multiplayer_mode($query, $count);

?>
```

Platform

```php
public function platform(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Platform (https://api-docs.igdb.com/#platform) endpoint.

Endpoint Description: The hardware used to run the game or game delivery network

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Platform endpoint method
    $igdb->platform($query, $count);

?>
```

Language Support Type

```
public function language_support_type(string $query, boolean $count = false)
throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Language Support Type (https://api-docs.igdb.com/#language-support-type) endpoint.

Endpoint Description: Language Support Types contains the identifiers for the support types that Language Support uses.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Language Support Type endpoint method
    $igdb->language_support_type($query, $count);

?>
```

Platform Family

```
public function platform_family(string $query, boolean $count = false) throws
IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Platform Family (https://api-docs.igdb.com/#platform-family) endpoint.

Endpoint Description: A collection of closely related platforms

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Platform Family endpoint method
    $igdb->platform_family($query, $count);

?>
```

Network Type

```
public function network_type(string $query, boolean $count = false) throws
IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Network Type (https://api-docs.igdb.com/#network-type) endpoint.

Endpoint Description: Social networks related to the event like twitter, facebook and youtube

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Network Type endpoint method
    $igdb->network_type($query, $count);

?>
```

Platform Logo

```
public function platform_logo(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Platform Logo (https://api-docs.igdb.com/#platform-logo) endpoint.

Endpoint Description: Logo for a platform

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Platform Logo endpoint method
    $igdb->platform_logo($query, $count);

?>
```

Platform Version Company

```
public function platform_version_company(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Platform Version Company (https://api-docs.igdb.com/#platform-version-company) endpoint.

Endpoint Description: A platform developer

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Platform Version Company endpoint method
    $igdb->platform_version_company($query, $count);

?>
```

Platform Version

```php
public function platform_version(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Platform Version (https://api-docs.igdb.com/#platform-version) endpoint.

Endpoint Description:

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Platform Version endpoint method
    $igdb->platform_version($query, $count);

?>
```

Platform Website

```php
public function platform_website(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Platform Website (https://api-docs.igdb.com/#platform-website) endpoint.

Endpoint Description: The main website for the platform

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Platform Website endpoint method
    $igdb->platform_website($query, $count);

?>
```

Platform Version Release Date

```
public function platform_version_release_date(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Platform Version Release Date (https://api-docs.igdb.com/#platform-version-release-date) endpoint.

Endpoint Description: A handy endpoint that extends platform release dates. Used to dig deeper into release dates, platforms and versions.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Platform Version Release Date endpoint method
    $igdb->platform_version_release_date($query, $count);

?>
```

Player Perspective

```
public function player_perspective(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Player Perspective (https://api-docs.igdb.com/#player-perspective) endpoint.

Endpoint Description: Player perspectives describe the view/perspective of the player in a video game.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Player Perspective endpoint method
    $igdb->player_perspective($query, $count);

?>
```

Region

```
public function region(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Region (https://api-docs.igdb.com/#region) endpoint.

Endpoint Description: Region for game localization

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Region endpoint method
    $igdb->region($query, $count);

?>
```

Release Date

```
public function release_date(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Release Date (https://api-docs.igdb.com/#release-date) endpoint.

Endpoint Description: A handy endpoint that extends game release dates. Used to dig deeper into release dates, platforms and versions.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Release Date endpoint method
    $igdb->release_date($query, $count);

?>
```

Release Date Status

```
public function release_date_status(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Release Date Status (https://api-docs.igdb.com/#release-date-status) endpoint.

Endpoint Description: An endpoint to provide definition of all of the current release date statuses.

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Release Date Status endpoint method
    $igdb->release_date_status($query, $count);

?>
```

Screenshot

```php
public function screenshot(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Screenshot (https://api-docs.igdb.com/#screenshot) endpoint.

Endpoint Description: Screenshots of games

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Screenshot endpoint method
    $igdb->screenshot($query, $count);

?>
```

Search

```php
public function search(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Search (https://api-docs.igdb.com/#search) endpoint.

Endpoint Description:

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```php
<?php

    // Search endpoint method
    $igdb->search($query, $count);

?>
```

Theme

```
public function theme(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Theme (https://api-docs.igdb.com/#theme) endpoint.

Endpoint Description: Video game themes

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Theme endpoint method
    $igdb->theme($query, $count);

?>
```

Website

```
public function website(string $query, boolean $count = false) throws IGDBEndpointException: mixed
```

Fetching data from IGDB API using the Website (https://api-docs.igdb.com/#website) endpoint.

Endpoint Description: A website url, usually associated with a game

Parameters:

- `$query`: an apicalypse formatted query String
- `$count`: whether the request should return the number of matches instead of the actual resultset

Returns: either the resultset as an array of objects, or a single object with a count property. Depends on the second `$count` parameter.

```
<?php

    // Website endpoint method
    $igdb->website($query, $count);

?>
```

MultiQuery

```
public function multiquery(array $queries) throws IGDBEndpointException, IGDBInvalidParameterException: mixed
```

This method executes a query against the `multiquery` endpoint. With this functionality one is able to execute multiple queries in a single request.

:warning If you are using the Query Builder to construct your queries, the parameters `name` and `endpoint` are **mandatory**! There is also a third optional parameter `count`. If any of the mandatory parameters are missing for the multiquery, an `IGDBInvalidParameterException` is thrown! Please refer to the `build` method for more information!

Parameters

- `$queries`: an array of apicalypse formatted multiquery query strings.

Returns: the response from the IGDB endpoint. The result object can vary depending on your query.

```php
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the wrapper
    $igdb = new IGDB("{client_id}", "{access_token}");

    // query builder for the main game
    $main_builder = new IGDBQueryBuilder();

    // query builder for the bundles
    $bundle_builder = new IGDBQueryBuilder();

    try {
        // building the main game query
        $main = $main_builder
            ->name("Main Game")
            ->endpoint("game")
            ->fields("id,name")
            ->where("id = 25076")
            ->build(true);

        // building the bundle query
        $bundle = $bundle_builder
            ->name("Bundles")
            ->endpoint("game")
            ->fields("id,name,version_parent,category")
            ->where("version_parent = 25076")
            ->where("category = 0")
            ->build(true);

        // the query can be passed as a string too

        // $main = "query games \"Main Game\" {
        //     fields id,name;
        //     where id = 25076;
        // };";

        // $bundle = "query games \"Bundles\" {
        //     fields id,name,version_parent,category;
        //     where version_parent = 25076 & category = 0;
        // };";
```

```
        // passing the queries to the multiquery method as an array of strings
        var_dump(
            $igdb->multiquery(
                array($main, $bundle)
            )
        );
    } catch (IGDBInvaliParameterException $e) {
        // a builder property is invalid
        echo $e->getMessage();
    } catch (IGDBEndpointException $e) {
        // something went wront with the query
        echo $e->getMessage();
    }
?>
```

The result of the query:

```
array (size=2)
  0 =>
    object(stdClass)[4]
      public 'name' => string 'Main Game' (length=9)
      public 'result' =>
        array (size=1)
          0 =>
            object(stdClass)[5]
              public 'id' => int 25076
              public 'name' => string 'Red Dead Redemption 2' (length=21)
  1 =>
    object(stdClass)[6]
      public 'name' => string 'Bundles' (length=7)
      public 'result' =>
        array (size=3)
          0 =>
            object(stdClass)[7]
              public 'id' => int 103207
              public 'category' => int 0
              public 'name' => string 'Red Dead Redemption 2: Collector's Box'
(length=38)
              public 'version_parent' => int 25076
          1 =>
            object(stdClass)[8]
              public 'id' => int 103206
              public 'category' => int 0
              public 'name' => string 'Red dead Redemption 2: Ultimate Edition'
(length=39)
              public 'version_parent' => int 25076
          2 =>
            object(stdClass)[9]
              public 'id' => int 103205
              public 'category' => int 0
              public 'name' => string 'Red Dead Redemption 2: Special Edition'
(length=38)
              public 'version_parent' => int 25076
```

Return Values

Every endpoint method can return two different type of results, depending on the second parameter passed to them.

By default the second `$count` parameter is a boolean `false`. this means, that the query will be executed against the IGDB, returning a `$result` **array of objects**.

```
<?php

    // a query against the game endpoint without a $count parameter
    $igdb->game("fields id,name; where id = (1,2);");

?>
```

The result of the query above:

```
array (size=2)
    0 =>
        object(stdClass)[2]
        public 'id' => int 1
        public 'name' => string 'Thief II: The Metal Age' (length=23)
    1 =>
        object(stdClass)[3]
        public 'id' => int 2
        public 'name' => string 'Thief: The Dark Project' (length=23)
```

If you pass a boolean `true` as a second parameter, then you will get an object with a `count` property containing the item count from the selected endpoint filtered by the `$query` filters.

```
<?php
    // a query against the game endpoint with a second true parameter
    // note the second boolean true parameter
    $igdb->game("fields id,name; where id = (1,2);", true);

?>
```

The result of the query above:

```
object(stdClass)[3]
    public 'count' => int 2
```

The result object's properties will vary depending on the provided field list in the `$query`. From the example result above you can see, the result holds an array, containing two elements. Every element of the result array is an object, containing properties with name of the fields from the `fields` parameter.

IGDB Utils

Utility tools for making the job easier with the IGDB API

Authenticate

```
public static function authenticate(string $client_id, string $client_secret)
throws Exception: object
```

A helper method to generate your `access_token`. This method will send a post request to the Twitch API with your `client_id` and `client_secret`.

Parameters:

- `$client_id`: your client id
- `$client_secret`: the client secret generated by twitch

Retunrs: an object decoded from the response from the Twitch API containing three keys: `access_token`, `expires_in` and `token_type`.

If a non-successful response is recieved from Twitch, an `Exception` is thrown.

```
<?php

    require_once "class.igdb.php";

    try {
        var_dump(IGDBUtils::authenticate("{client_id}", "{client_secret}"));
    } catch (Exception $e) {
        // something went wrong
        echo $e->getMessage();
    }

?>
```

The output of the method will contain your new `access_token`:

```
object(stdClass)[1]
  public 'access_token' => string 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx' (length=30)
  public 'expires_in' => int 5269900
  public 'token_type' => string 'bearer' (length=6)
```

Image URL

```
public static function image_url(string $image_id, string $size) throws
IGDBInvalidParameterException: string
```

Get the URL of a specific sized image for a record.

Parameters:

- `$image_id`: the `image_id` of the image. Each image in the database has it's own `id` and `image_id` fields. You **must** use the `image_id` field!
- `$size`: the required size. Possible sizes:
    - cover_small
    - cover_small_2x
    - screenshot_med
    - screenshot_med_2x
    - cover_big
    - cover_big_2x
    - logo_med
    - logo_med_2x
    - screenshot_big
    - screenshot_big_2x

- screenshot_huge
- screenshot_huge_2x
- thumb
- thumb_2x
- micro
- micro_2x
- 720p
- 720p_2x
- 1080p
- 1080p_2x

If an invalid second parameter is passed, an `IGDBInvalidParameterException` will be thrown.

```
<?php

    require_once "class.igdb.php";

    $igdb = new IGDB("{client_id}", "{access_token}");
    $builder = new IGDBQueryBuilder();

    try {
        $result = $igdb->game(
            $builder
                ->fields("id, name, artworks.*")
                ->search("uncharted 4")
                ->limit(1)
                ->build()
        );

        var_dump($result);

        // processing the first result's first artwork
        echo "720p version: " . IGDBUtils::image_url($result[0]->artworks[0]-
>image_id, "720p");
    } catch (IGDBInvalidParameterException $e) {
        // invalid parameter passed to the builder
        echo $e->getMessage();
    } catch (IGDBEndpointException $e) {
        // failed query
        echo $e->getMessage();
    }

?>
```

The script above will produce the output below:

```
array (size=1)
  0 =>
    object(stdClass)[3]
      public 'id' => int 7331
      public 'artworks' =>
        array (size=5)
          0 =>
            object(stdClass)[4]
              public 'id' => int 6303
```

```
              public 'alpha_channel' => boolean false
              public 'animated' => boolean false
              public 'game' => int 7331
              public 'height' => int 1059
              public 'image_id' => string 'ar4v3' (length=5)
              public 'url' => string
'//images.igdb.com/igdb/image/upload/t_thumb/ar4v3.jpg' (length=53)
              public 'width' => int 1500
              public 'checksum' => string 'ded76617-94df-a813-28cb-
feffebeb9d36' (length=36)
          1 =>
            object(stdClass)[5]
              public 'id' => int 6304
              public 'alpha_channel' => boolean false
              public 'animated' => boolean false
              public 'game' => int 7331
              public 'height' => int 1059
              public 'image_id' => string 'ar4v4' (length=5)
              public 'url' => string
'//images.igdb.com/igdb/image/upload/t_thumb/ar4v4.jpg' (length=53)
              public 'width' => int 1500
              public 'checksum' => string '5655acde-a94d-8aa5-8c35-
2ef4bc2e965d' (length=36)
          2 =>
            object(stdClass)[6]
              public 'id' => int 6305
              public 'alpha_channel' => boolean false
              public 'animated' => boolean false
              public 'game' => int 7331
              public 'height' => int 1060
              public 'image_id' => string 'ar4v5' (length=5)
              public 'url' => string
'//images.igdb.com/igdb/image/upload/t_thumb/ar4v5.jpg' (length=53)
              public 'width' => int 1500
              public 'checksum' => string '30441489-b264-70d7-2d26-
86b7c1eb3d3a' (length=36)
          3 =>
            object(stdClass)[7]
              public 'id' => int 6306
              public 'alpha_channel' => boolean false
              public 'animated' => boolean false
              public 'game' => int 7331
              public 'height' => int 1060
              public 'image_id' => string 'ar4v6' (length=5)
              public 'url' => string
'//images.igdb.com/igdb/image/upload/t_thumb/ar4v6.jpg' (length=53)
              public 'width' => int 1500
              public 'checksum' => string 'e8629a83-2452-ec8c-2388-
f42c6ec21de5' (length=36)
          4 =>
            object(stdClass)[8]
              public 'id' => int 6307
              public 'alpha_channel' => boolean false
              public 'animated' => boolean false
              public 'game' => int 7331
```

```
              public 'height' => int 1060
              public 'image_id' => string 'ar4v7' (length=5)
              public 'url' => string
'//images.igdb.com/igdb/image/upload/t_thumb/ar4v7.jpg' (length=53)
              public 'width' => int 1500
              public 'checksum' => string 'e0b49da1-76dd-a863-b499-
30b7f0ae0c78' (length=36)
      public 'name' => string 'Uncharted 4: A Thief's End' (length=26)
720p version: https://images.igdb.com/igdb/image/upload/t_720p/ar4v3.jpg
```

On the last line you can see the 720p version of the image which is a valid url.

Webhooks

List, create, update and delete webhooks with the Webhook Util.

Examples

Working examples, covering most of the functionalities of the wrapper

Basic Example

A basic example to send your apicalypse query to the IGDB API.

> Make sure to place your endpoint method calls in a try...catch block to be able to catch errors!

Code

```php
<?php

    // importing the wrapper
    require 'class.igdb.php';

    // instantiating the wrapper
    $igdb = new IGDB("{client_id}", "{access_token}");

    // your query string
    $query = 'search "uncharted"; fields id,name,cover; limit 5; offset 10;';

    try {
        // executing the query
        $games = $igdb->game($query);

        // showing the results
        var_dump($games);
    } catch (IGDBEnpointException $e) {
        // a non-successful response recieved from the IGDB API
        echo $e->getMessage();
    }

?>
```

Result

```
array (size=5)
  0 =>
    object(stdClass)[3]
      public 'id' => int 125062
      public 'cover' => int 83686
      public 'name' => string 'Uncharted Ocean: Set Sail' (length=25)
  1 =>
    object(stdClass)[4]
      public 'id' => int 19583
      public 'cover' => int 15883
      public 'name' => string 'Uncharted: Fight for Fortune' (length=28)
  2 =>
    object(stdClass)[5]
      public 'id' => int 26193
      public 'cover' => int 85149
      public 'name' => string 'Uncharted: The Lost Legacy' (length=26)
  3 =>
    object(stdClass)[6]
      public 'id' => int 19609
      public 'cover' => int 85164
      public 'name' => string 'Uncharted: Fortune Hunter' (length=25)
  4 =>
    object(stdClass)[7]
      public 'id' => int 7331
      public 'cover' => int 81917
      public 'name' => string 'Uncharted 4: A Thief's End' (length=26)
```

Using the Query Builder

An example to see how to use the IGDB Query Builder to build the query strings.

> Make sure to place your query builder configuration and endpoint method calls in a try...catch block to be able to catch errors!

Code

```php
<?php

    // importing the wrapper
    require 'class.igdb.php';

    // instantiating the wrapper
    $igdb = new IGDB("{client_id}", "{access_token}");

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // building the query
        $query = $builder
            // searching for games LIKE uncharted
            ->search("uncharted")
            // we want to see these fields in the results
            ->fields("id, name, cover")
            // we only need maximum 5 results per query (pagination)
            ->limit(5)
            // we would like to show the third page; fetch the results from the
tenth element (pagination)
            ->offset(10)
            // process the configuration and return a string
            ->build();

        // executing the query
        $games = $igdb->game($query);

        // showing the results
        var_dump($games);
    } catch (IGDBInvalidParameterException $e) {
        // an invalid parameter is passed to the query builder
        echo $e->getMessage();
    } catch (IGDBEnpointException $e) {
        // a non-successful response recieved from the IGDB API
        echo $e->getMessage();
    }

?>
```

Result

```
array (size=5)
  0 =>
    object(stdClass)[3]
      public 'id' => int 125062
      public 'cover' => int 83686
      public 'name' => string 'Uncharted Ocean: Set Sail' (length=25)
  1 =>
    object(stdClass)[4]
      public 'id' => int 19583
      public 'cover' => int 15883
      public 'name' => string 'Uncharted: Fight for Fortune' (length=28)
  2 =>
    object(stdClass)[5]
      public 'id' => int 26193
      public 'cover' => int 85149
      public 'name' => string 'Uncharted: The Lost Legacy' (length=26)
  3 =>
    object(stdClass)[6]
      public 'id' => int 19609
      public 'cover' => int 85164
      public 'name' => string 'Uncharted: Fortune Hunter' (length=25)
  4 =>
    object(stdClass)[7]
      public 'id' => int 7331
      public 'cover' => int 81917
      public 'name' => string 'Uncharted 4: A Thief's End' (length=26)
```

Query Builder with Options

The IGDB Query Builder still supports the legacy `$options` array to parameterize the query.

> :warning Using the Builder this way is not recommended as this functionality may be removed in future versions. Use the builder approach instead.

Code

```php
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiating the wrapper
    $igdb = new IGDB("{client_id}", "{access_token}");

    // instantiating the builder
    $builder = new IGDBQueryBuilder();

    // creating the options array
    $options = array(
        // searching for games LIKE uncharted
        "search" => "uncharted",
        // we want to see these fields in the results
        "fields" => array("id", "name", "cover"),
        // we only need maximum 5 results per query (pagination)
        "limit" => 5,
        // we would like to show the third page; fetch the results from the
tenth element (pagination)
        "offset" => 10
    );

    try {
        // adding your $options array with the options method
        $builder->options($options);

        // building the query
        $query = $builder->build();

        // executing the query
        $games = $igdb->game($query);

        // showing the results
        var_dump($games);
    } catch (IGDBInvalidParameterException $e) {
        // an invalid parameter is passed to the query builder
        echo $e->getMessage();
    } catch (IGDBEnpointException $e) {
        // a non-successful response recieved from the IGDB API
        echo $e->getMessage();
    }

?>
```

Result

```
array (size=5)
  0 =>
    object(stdClass)[3]
      public 'id' => int 125062
      public 'cover' => int 83686
      public 'name' => string 'Uncharted Ocean: Set Sail' (length=25)
  1 =>
    object(stdClass)[4]
      public 'id' => int 19583
      public 'cover' => int 15883
      public 'name' => string 'Uncharted: Fight for Fortune' (length=28)
  2 =>
    object(stdClass)[5]
      public 'id' => int 26193
      public 'cover' => int 85149
      public 'name' => string 'Uncharted: The Lost Legacy' (length=26)
  3 =>
    object(stdClass)[6]
      public 'id' => int 19609
      public 'cover' => int 85164
      public 'name' => string 'Uncharted: Fortune Hunter' (length=25)
  4 =>
    object(stdClass)[7]
      public 'id' => int 7331
      public 'cover' => int 81917
      public 'name' => string 'Uncharted 4: A Thief's End' (length=26)
```

Counting Results

An example to count the matched records.

> When `true` is passed as the second parameter, the return value will be an object with a single property called `count`. For more details on the return values of the endpoint methods please refer to the return values section.

Code

```php
<?php

    // importing the wrapper
    require 'class.igdb.php';

    // instantiating the wrapper
    $igdb = new IGDB("{client_id}", "{access_token}");

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // building the query
        $query = $builder
            // setting a filter to fetch games with rating greater than 75
            ->where(
                array(
                    'field' => 'rating',
                    'postfix' => '>',
                    'value' => 75
                )
            )
            // process the configuration and return a string
            ->build();

        // executing the query
        // note the second true parameter
        $game_count = $igdb->game($query, true);

        // showing the results
        var_dump($game_count);
    } catch (IGDBInvalidParameterException $e) {
        // an invalid parameter is passed to the query builder
        echo $e->getMessage();
    } catch (IGDBEnpointException $e) {
        // a non-successful response recieved from the IGDB API
        echo $e->getMessage();
    }

?>
```

Result

```
object(stdClass)[3]
  public 'count' => int 8081
```

Expander

Some fields are actually ids pointing to other endpoints. The expander feature is a convenient way to go into these other endpoints and access more information from them in the same query, instead of having to do multiple queries.

Code

```php
<?php

    // importing the wrapper
    require 'class.igdb.php';

    // instantiating the wrapper
    $igdb = new IGDB("{client_id}", "{access_token}");

    // instantiate the query builder
    $builder = new IGDBQueryBuilder();

    try {
        // building the query
        $query = $builder
            // fetching the first 2 games by id 1 and 2
            ->id(array(1,2))
            // fields can be expanded with a dot followed by the fields you
want to access from a certain endpoint
            ->fields(array("name", "themes.url", "themes.name"))
            // process the configuration and return a string
            ->build();

        // executing the query
        $game_count = $igdb->game($query);

        // showing the results
        var_dump($game_count);
    } catch (IGDBInvalidParameterException $e) {
        // an invalid parameter is passed to the query builder
        echo $e->getMessage();
    } catch (IGDBEnpointException $e) {
        // a non-successful response recieved from the IGDB API
        echo $e->getMessage();
    }

?>
```

Result

```
array (size=2)
  0 =>
    object(stdClass)[3]
      public 'id' => int 1
      public 'name' => string 'Thief II: The Metal Age' (length=23)
      public 'themes' =>
        array (size=3)
          0 =>
            object(stdClass)[4]
              public 'id' => int 1
              public 'name' => string 'Action' (length=6)
              public 'url' => string 'https://www.igdb.com/themes/action'
(length=34)
          1 =>
            object(stdClass)[5]
              public 'id' => int 17
              public 'name' => string 'Fantasy' (length=7)
              public 'url' => string 'https://www.igdb.com/themes/fantasy'
(length=35)
          2 =>
            object(stdClass)[6]
              public 'id' => int 23
              public 'name' => string 'Stealth' (length=7)
              public 'url' => string 'https://www.igdb.com/themes/stealth'
(length=35)
  1 =>
    object(stdClass)[7]
      public 'id' => int 2
      public 'name' => string 'Thief: The Dark Project' (length=23)
      public 'themes' =>
        array (size=3)
          0 =>
            object(stdClass)[8]
              public 'id' => int 1
              public 'name' => string 'Action' (length=6)
              public 'url' => string 'https://www.igdb.com/themes/action'
(length=34)
          1 =>
            object(stdClass)[9]
              public 'id' => int 17
              public 'name' => string 'Fantasy' (length=7)
              public 'url' => string 'https://www.igdb.com/themes/fantasy'
(length=35)
          2 =>
            object(stdClass)[10]
              public 'id' => int 23
              public 'name' => string 'Stealth' (length=7)
              public 'url' => string 'https://www.igdb.com/themes/stealth'
(length=35)
```

MultiQuery

Using multiquery multiple queries can be executed against the IGDB database using a single query. The multiquery method expects an array of multiquery query strings.

:info Using the `build` method with a boolean `true` parameter, a query will be returned with a multiquery syntax.

Code

```php
<?php

    // importing the wrapper
    require_once "class.igdb.php";

    // instantiate the wrapper
    $igdb = new IGDB("{client_id}", "{access_token}");

    // query builder for the main game
    $main = new IGDBQueryBuilder();

    // query builder for the bundles
    $bundle = new IGDBQueryBuilder();

    try {
        // configuring the main query
        $main
            ->name("Main Game")
            ->endpoint("game")
            ->fields("id,name")
            ->where("id = 25076");

        // configuring the bundle query
        $bundle
            ->name("Bundles")
            ->endpoint("game")
            ->fields("id,name,version_parent,category")
            ->where("version_parent = 25076")
            ->where("category = 0");

        var_dump(
            $igdb->multiquery(
                array(
                    $main->build(true),
                    $bundle->build(true)
                )
            )
        );
    } catch (IGDBInvaliParameterException $e) {
        // a builder property is invalid
        echo $e->getMessage();
    } catch (IGDBEndpointException $e) {
        // something went wront with the query
        echo $e->getMessage();
    }
?>
```

Result

```
array (size=2)
  0 =>
    object(stdClass)[4]
      public 'name' => string 'Main Game' (length=9)
      public 'result' =>
        array (size=1)
          0 =>
            object(stdClass)[5]
              public 'id' => int 25076
              public 'name' => string 'Red Dead Redemption 2' (length=21)
  1 =>
    object(stdClass)[6]
      public 'name' => string 'Bundles' (length=7)
      public 'result' =>
        array (size=3)
          0 =>
            object(stdClass)[7]
              public 'id' => int 103205
              public 'category' => int 0
              public 'name' => string 'Red Dead Redemption 2: Special Edition'
(length=38)
              public 'version_parent' => int 25076
          1 =>
            object(stdClass)[8]
              public 'id' => int 103207
              public 'category' => int 0
              public 'name' => string 'Red Dead Redemption 2: Collector's Box'
(length=38)
              public 'version_parent' => int 25076
          2 =>
            object(stdClass)[9]
              public 'id' => int 103206
              public 'category' => int 0
              public 'name' => string 'Red dead Redemption 2: Ultimate Edition'
(length=39)
              public 'version_parent' => int 25076
```

Change Log

Changes, updates, notes.

v4.3.2 - October 26, 2023

- Added new endpoint methods to the wrapper
    - collection_membership
    - collection_membership_type
    - collection_relation
    - collection_relation_type
    - collection_type
    - event
    - event_logo
    - event_network
    - game_localization

    - language
    - language_support
    - language_support_type
    - network_type
    - region
    - release_date_status

v4.3.1 - April 19, 2022

- IGDBEndpointException
    - Added a `getResponseCode()` to fetch http response code from IGDB API
- Added sections to the documentation:
    - Handling Errors
    - Handling Builder Errors
    - Handling Request Errors

v4.3.0 - August 19, 2021

- IGDBQueryBuilder: three new properties introduced for multiquery:
    - name
    - endpoint
    - count
- IGDBQueryBuilder: build method signature updated
- IGDBWrapper: multiquery updated to accept array of queries

v4.2.0 - May 22, 2021

- IGDBQueryBuilder: Moved the `$options` array parsing to the `options()` method
- IGDBQueryBuilder: `reset()` method added

v4.1.2 - May 20, 2021

- Minor updates to the readme

v4.1.1 - May 20, 2021

- Removed a debugging var_dump from the IGDB Wrapper
- Updated the documentation with a Query Builder example with `$options` array

v4.1.0 - May 15, 2021

- The wrapper got a brand new documentation!
- Introduced the IGDBQueryBuilder class
- Introduced the IGDB Utils class
- Introduced `IGDBEndpointException` and `IGDBInvalidParameterException` classes
- The wrapper endpoint methods no longer accepts `$options`, only apicalypse query strings (https://api-docs.igdb.com/#apicalypse-1)

v4.0.2 - April 28, 2021

- Updated error response handling from IGDB

v4.0.1 - February 18, 2021

- Minor updates to the Readme

v4.0.0 - October 20, 2020

- **IGDB Api v4 compatibility update**
- Updated Class constructor to accept the new tokens from Twitch
- Removed API KEY
- Removed `IGDB::api_status()` method
- Removed Endpoint methods according to the IGDB Changes (https://api-docs.igdb.com/#breaking-changes)
- Renamed methods:
    - `_init_curl() => _curl_init()`
    - `close_curl() => curl_close()`
    - `reinit_curl() => curl_reinit()`
- Updated endpoint methods to accept apicalypse strings as well
- Implemented Multiquery

v2.0.3 - September 17, 2020

- Fixed a bug with the `where` filter (#6 Issues with slug field (https://github.com/enisz/igdb/issues/6))

v2.0.2 - February 03, 2020

- Fixing inaccurate information in the Readme

v2.0.1 - January 27, 2020

- Minor changes / fixes in the Readme
- Added method `_construct_url`
- Updated every endpoint method to construct the endpoint url's different

v2.0.0 - December 04, 2019

- **IGDB Api v3 compatibility update**
- Removed `expander` parameter
- Renamed parameter `filter` to `where`
- Renamed parameter `order` to `sort`
- Removed multiple methods:
    - `_stringify_options`
    - `_construct_url`
    - `count`
    - `custom_query`
- Added method `apicalypse`
- Added method `api_status`

- Updated every endpoint method (removed `$execute`, added `$count`)

v1.0.5 - March 11, 2019

- Fixed a bug at the request's error handling
- `public IGDB::get_request_info()` public method added

v1.0.4 - March 25, 2018

- Default properties has been removed.
- set_default public method has been removed.

v1.0.3 - March 18, 2018

- Providing either search or id parameter in the options array are not mandatory anymore.
- Providing fields parameter when using expander is not mandatory anymore.
- Ordering parameter 'order' in the options array has been renamed to 'direction'. Refer to the order section of the options parameters.
- Implemented count method. Refer to the count section of the Readme.
- Example *count.php* has been added.
- Updated Readme

v1.0.2 - March 17, 2018

- Modified the constructor to ask only for the API Key. The API URL has been changed to be fix for every user (by IGDB).
- The API URL and KEY setter and getter methods has been removed.
- The API URL and KEY validator methods has been removed.
- New method for order parameter constructing has been implemented.
- Stringify Options method is private again. Use the updated endpoint methods instead.
- Updated Endpoint Methods to accept a second optional parameter to return the constructed URL instead of executing the query.
- *basic.php* example file has been renamed to *basic.example.php*.
- *order.php* example has been added.
- *order_subfilter.php* example has been added.
- All example files has been modified with the updated constructor.

v1.0.1 - March 16, 2018

- Added Changes section to the ReadMe.
- Fixed filter parameter constructing; the parameter input has been changed.
- Added example snippets to the Options Parameters section.
- Added example file *filter_multiple_criteria.php*
- Added example file *filter_single_criteria.php*

test

asdf

test

asdf
sdf

test

asfd
asdf

test

léghwerg
wew

test

ewrtq
qewr

test

qwer
qwr

test

askfsadjf
asjdfasdkfj