
Firestore Admin SDK for PHP

Release

Aug 11, 2017

1	User Guide	3
1.1	Overview	3
1.1.1	Requirements	3
1.1.2	Installation	3
1.1.3	Issues/Support	4
1.1.4	Roadmap	4
1.1.5	License	4
1.1.6	Contributing	5
1.1.7	Acknowledgements	5
1.2	Setup	5
1.2.1	Google Service Account	5
1.2.2	Custom Database URI	7
1.3	Realtime Database	7
1.3.1	Retrieving data	7
1.3.2	Saving data	11
1.4	Authentication	12
1.4.1	Impersonating users	12
1.4.2	Working with Tokens	13
1.5	Troubleshooting	13
1.5.1	Call to undefined function openssl_sign()	13
1.5.2	cURL error XX: SSL certificate validation failed	14
1.6	Migration	14
1.6.1	3.0 to 3.1	14
1.6.2	2.x to 3.x	15

Interact with Google Firebase from your PHP application.

The source code can be found at <https://github.com/kreait/firebase-php/>

```
<?php

require __DIR__ . '/vendor/autoload.php';

use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;

$serviceAccount = ServiceAccount::fromJsonFile(__DIR__ . '/google-service-account.json'
↪);

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->withDatabaseUri('https://my-project.firebaseio.com')
    ->create();

$database = $firebase->getDatabase();

$newPost = $database
    ->getReference('blog/posts')
    ->push([
        'title' => 'Post title',
        'body' => 'This should probably be longer.'
    ]);

$newPost->getKey(); // => -Kvr5eu8gcTv7_AHb-3-
$newPost->getUri(); // => https://my-project.firebaseio.com/blog/posts/-Kvr5eu8gcTv7_
↪AHb-3-

$newPost->getChild('title')->set('Changed post title');
$newPost->getValue(); // Fetches the data from the realtime database
$newPost->remove();
```


Overview

Requirements

- PHP \geq 7.0
- The [mbstring](#) PHP extension
- A Firebase project - create a new project in the [Firebase console](#), if you don't already have one.
- A Google service account, follow the instructions in the [official Firebase Server documentation](#) and place the JSON configuration file somewhere in your project's path.

Installation

The recommended way to install the Firebase Admin SDK is with [Composer](#). Composer is a dependency management tool for PHP that allows you to declare the dependencies your project needs and installs them into your project.

```
# Install Composer
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

You can add the Firebase Admin SDK as a dependency using the composer.phar CLI:

```
php composer.phar require kreait/firebase-php ^3.0
```

Alternatively, you can specify the Firebase Admin SDK as a dependency in your project's existing composer.json file:

```
{
  "require": {
    "kreait/firebase-php": "^3.0"
  }
}
```

```
}  
}
```

After installing, you need to require Composer’s autoloader:

```
<?php  
  
require __DIR__ . '/vendor/autoload.php';
```

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

Issues/Support

- [Github issue tracker](#)
- Join the Firebase Community Slack at <https://firebase-community.appspot.com>, join the #php channel and look for @jeromegamez.

Roadmap

The following planned features are not in a particular order:

- Integration of [Firestore Storage](#)
- Automatic updates of [Firestore Rules](#)
 - Background: [Data must be indexed to be queriable or sortable](#). If you try to query a yet unindexed dataset, the Firestore REST API will return an error. With this feature, the SDK could execute an error, and if an error occurs, update the Firestore Rules as needed and retry.
- Support for listening to the [Firestore event stream](#)
- PHP Object Serialization and Deserialization
- Use parallel requests where possible to speed up operations

License

Licensed using the [MIT license](#).

Copyright (c) 2016-2017 Jérôme Gamez <<https://github.com/jeromegamez>> <jerome@gamez.name>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR

OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Contributing

Guidelines

1. The SDK utilizes PSR-1, PSR-2, PSR-4, and PSR-7.
2. This SDK has a minimum PHP version requirement of PHP 7.0. Pull requests must not require a PHP version greater than PHP 7.0 unless the feature is only utilized conditionally.
3. All pull requests must include unit tests to ensure the change works as expected and to prevent regressions.

Running the tests

The SDK is unit tested with PHPUnit. Run the tests using the Makefile:

```
make tests
```

Coding standards

The SDK uses the [PHP Coding Standards Fixer](#) to ensure a uniform coding style. Apply coding standard fixed using the Makefile:

```
make cs
```

from the root of the project.

Acknowledgements

- The structure and wording of this documentation is loosely based on the official Firebase documentation at <https://firebase.google.com/docs/>.
- The index and overview page are adapted from [Guzzle's documentation](#).

Setup

Google Service Account

In order to access a Firebase project using a server SDK, you must authenticate your requests to Firebase with a [Service Account](#).

Follow the steps described in the official Firebase documentation to create a Service Account for your Firebase application (see [Add the Firebase Admin SDK to your Server](#)) and make sure the Service Account has the *Project -> Editor* or *Project -> Owner* role.

With autodiscovery

By default, the SDK is able to autodiscover the Service Account for your project in the following conditions:

1. The path to the JSON key file is defined in one of the following environment variables
 - FIREBASE_CREDENTIALS
 - GOOGLE_APPLICATION_CREDENTIALS
2. The JSON Key file is located in Google's "well known path"
 - on Linux/macOS: `$HOME/.config/gcloud/application_default_credentials.json`
 - on Windows: `$APPDATA/gcloud/application_default_credentials.json`

If one of the conditions above is met, creating a new Firebase instance is as easy as this:

```
use Krait\Firebase\Factory;

$firebase = (new Factory)->create();
```

A more explicit alternative:

```
use Krait\Firebase\Factory;
use Krait\Firebase\ServiceAccount;

$serviceAccount = ServiceAccount::discover();

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->create();
```

Manually

You can also pass the path to the Service Account JSON file explicitly:

```
use Krait\Firebase\Factory;
use Krait\Firebase\ServiceAccount;

$serviceAccount = ServiceAccount::fromJsonFile(__DIR__.'/firebase_credentials.json');
$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->create();
```

Use your own autodiscovery

You can use your own, custom autodiscovery methods as well:

```
use Krait\Firebase\Factory;
use Krait\Firebase\ServiceAccount\Discoverer

$discoverer = new Discoverer([
    function () {
        $serviceAccount = ...; // Instance of Krait\Firebase\ServiceAccount

        return $serviceAccount;
    }
]);
```

```

  });

  $firebase = (new Factory)
    ->withServiceAccountDiscoverer($myDiscoverer)
    ->create();

```

Custom Database URI

If the project ID in the JSON file does not match the URL of your Firebase application, or if you want to be explicit, you can configure the Factory like this:

```

use Krait\Firebase\Factory;

$firebase = (new Factory)
  ->withDatabaseUri('https://my-project.firebaseio.com')
  ->create();

```

Realtime Database

You can work with your Firebase application's Realtime Database by invoking the `getDatabase()` method of your Firebase instance:

```

use Krait\Firebase;

$firebase = (new Firebase\Factory())->create();
$database = $firebase->getDatabase();

```

Retrieving data

Every node in your database can be accessed through a Reference:

```

$reference = $database->getReference('path/to/child/location');

```

Note: Creating a reference does not result in a request to your Database. Requests to your Firebase applications are executed with the `getSnapshot()` and `getValue()` methods only.

You can then retrieve a Database Snapshot for the Reference or its value directly:

```

$snapshot = $reference->getSnapshot();

$value = $snapshot->getValue();
// or
$value = $reference->getValue();

```

Database Snapshots

Database Snapshots are immutable copies of the data at a Firebase Database location at the time of a query. They can't be modified and will never change.

```
$snapshot = $reference->getSnapshot();
$value = $snapshot->getValue();

$value = $reference->getValue(); // Shortcut for $reference->getSnapshot()->
    =>getValue();
```

Snapshots provide additional methods to work with and analyze the contained value:

- `exists()` returns true if the Snapshot contains any (non-null) data.
- `getChild()` returns another Snapshot for the location at the specified relative path.
- `getKey()` returns the key (last part of the path) of the location of the Snapshot.
- `getReference()` returns the Reference for the location that generated this Snapshot.
- `getValue()` returns the data contained in this Snapshot.
- `hasChild()` returns true if the specified child path has (non-null) data.
- `hasChildren()` returns true if the Snapshot has any child properties, i.e. if the value is an array.
- `numChildren()` returns the number of child properties of this Snapshot, if there are any.

Queries

You can use Queries to filter and order the results returned from the Realtime Database. Queries behave exactly like References. That means you can execute any method on a Query that you can execute on a Reference.

Note: You can combine every filter query with every order query, but not multiple queries of each type. Shallow queries are a special case: they can not be combined with any other query method.

Shallow queries

This is an advanced feature, designed to help you work with large datasets without needing to download everything. Set this to true to limit the depth of the data returned at a location. If the data at the location is a JSON primitive (string, number or boolean), its value will simply be returned.

If the data snapshot at the location is a JSON object, the values for each key will be truncated to true.

Detailed information can be found on [the official Firebase documentation page for shallow queries](#)

```
$db->getReference('currencies')
    // order the reference's children by their key in ascending order
->shallow()
->getSnapshot();
```

A convenience method is available to retrieve the key names of a reference's children:

```
$db->getReference('currencies')->getChildKeys(); // returns an array of key names
```

Ordering data

The official Firebase documentation explains [How data is ordered](#).

Data is always ordered in ascending order.

You can only order by one property at a time - if you try to order by multiple properties, e.g. by child and by value, an exception will be thrown.

By key

```
$db->getReference('currencies')
    // order the reference's children by their key in ascending order
->orderByKey()
->getSnapshot();
```

By value

Note: In order to order by value, you must define an index, otherwise the Firebase API will refuse the query.

```
{
  "currencies": {
    ".indexOn": ".value"
  }
}
```

```
$db->getReference('currencies')
    // order the reference's children by their value in ascending order
->orderByValue()
->getSnapshot();
```

By child

Note: In order to order by a child value, you must define an index, otherwise the Firebase API will refuse the query.

```
{
  "people": {
    ".indexOn": "height"
  }
}
```

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height' in
    ↪ ascending order
->orderByChild('height')
->getSnapshot();
```

Filtering data

To be able to filter results, you must also define an order.

limitToFirst

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // limits the result to the first 10 children (in this case: the 10 shortest
    ↪persons)
    // values for 'height')
    ->limitToFirst(10)
    ->getSnapshot();
```

limitToLast

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // limits the result to the last 10 children (in this case: the 10 tallest
    ↪persons)
    ->limitToLast(10)
    ->getSnapshot();
```

startAt

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // returns all persons taller than or exactly 1.68 (meters)
    ->startAt(1.68)
    ->getSnapshot();
```

endAt

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // returns all persons shorter than or exactly 1.98 (meters)
    ->endAt(1.98)
    ->getSnapshot();
```

equalTo

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // returns all persons being exactly 1.98 (meters) tall
    ->equalTo(1.98)
    ->getSnapshot();
```

Saving data

Set/replace values

For basic write operations, you can use `set()` to save data to a specified reference, replacing any existing data at that path. For example a configuration array for a website might be set as follows:

```
$db->getReference('config/website')
->set([
    'name' => 'My Application',
    'emails' => [
        'support' => 'support@domain.tld',
        'sales' => 'sales@domain.tld',
    ],
    'website' => 'https://app.domain.tld',
]);

$db->getReference('config/website/name')->set('New name');
```

Note: Using `set()` overwrites data at the specified location, including any child nodes.

Update specific fields¹

To simultaneously write to specific children of a node without overwriting other child nodes, use the `update()` method.

When calling `update()`, you can update lower-level child values by specifying a path for the key. If data is stored in multiple locations to scale better, you can update all instances of that data using data fan-out.

For example, in a blogging app you might want to add a post and simultaneously update it to the recent activity feed and the posting user's activity feed using code like this:

```
$uid = 'some-user-id';
$postData = [
    'title' => 'My awesome post title',
    'body' => 'This text should be longer',
];

// Create a key for a new post
$newPostKey = $db->getReference('posts')->push()->getKey();

$updates = [
    'posts/'.$newPostKey => $postData,
    'user-posts/'.$uid.'/'.$newPostKey => $postData,
];

$db->getReference() // this is the root reference
->update($updates);
```

Writing lists²

¹ This example and its description is the same as in the official documentation: [Update specific fields](#).

² This example and its description is the same as in the official documentation: [Append to a list of data](#).

Use the `push()` method to append data to a list in multiuser applications. The `push()` method generates a unique key every time a new child is added to the specified Firestore reference. By using these auto-generated keys for each new element in the list, several clients can add children to the same location at the same time without write conflicts. The unique key generated by `push()` is based on a timestamp, so list items are automatically ordered chronologically.

You can use the reference to the new data returned by the `push()` method to get the value of the child's auto-generated key or set data for the child. The `getKey()` method of a `push()` reference contains the auto-generated key.

```
$postData = [...];
$postRef = $db->getReference('posts')->push($postData);

$postKey = $postRef->getKey(); // The key looks like this: -KVquJHezVLF-lSye6Qg
```

Server values

Server values can be written at a location using a placeholder value which is an object with a single `.sv` key. The value for that key is the type of server value you wish to set.

Firestore currently supports only one server value: `timestamp`. You can either set it manually in your write operation, or use a constant from the `Firestore\Database` class.

The following two usages are equivalent:

```
$ref = $db->getReference('posts/my-post')
    ->set('created_at', ['.sv' => 'timestamp']);

$ref = $db->getReference('posts/my-post')
    ->set('created_at', Database::SERVER_TIMESTAMP);
```

Delete data³

The simplest way to delete data is to call `remove()` on a reference to the location of that data.

```
$db->getReference('posts')->remove();
```

You can also delete by specifying `null` as the value for another write operation such as `set()` or `update()`.

```
$db->getReference('posts')->set(null);
```

You can use this technique with `update()` to delete multiple children in a single API call.

Authentication

Impersonating users

You can impersonate users of your Firestore application through the `asUserWithClaims()` method, which requires a user id as the first parameter, and an optional array with claims as the second.

```
use Kreait\Firebase;

$firebase = (new Firebase\Factory())->create();
```

³ This example and its description is the same as in the official documentation: [Delete data](#).


```
$authenticated = $firebase->asUserWithClaims('a-user-id', [
    'premium-user' => true
]);
```

Working with Tokens

If you need to create [Custom Tokens](#) or verify [ID Tokens](#), a Service Account based Firebase instance provides the `getTokenHandler()` method:

```
use Kreait\Firebase;

$firebase = (new Firebase\Factory())->create();

$tokenHandler = $firebase->getTokenHandler();

$uid = 'a-uid';
$claims = ['foo' => 'bar']; // optional

// Returns a Lcobucci\JWT\Token instance
$customToken = $tokenHandler->createCustomToken($uid, $claims);
echo $customToken; // "eyJ0eXAiOiJKV1..."

$tokenIdString = 'eyJhbGciOiJSUzI1...';
// Returns a Lcobucci\JWT\Token instance
$idToken = $tokenHandler->verifyIdToken($tokenIdString);

$uid = $idToken->getClaim('sub');

echo $uid; // 'a-uid'
```

If you want to use a custom token handler, you can do so by passing it to the factory:

```
use Kreait\Firebase;

$handler = new Firebase\Auth\Token\Handler(...);

$firebase = (new Firebase\Factory())
    ->withTokenHandler($handler);
    ->create();
```

Note: A standalone version of the Token Handler is available with the [kreait/firebase-tokens](#) library.

Troubleshooting

Call to undefined function openssl_sign()

You need to install the OpenSSL PHP Extension: <http://php.net/openssl>

cURL error XX: SSL certificate validation failed

If you receive the above error, make sure that you have a current CA Root Certificates bundle on your system and that PHP uses it.

To see where PHP looks for the CA bundle, check the output of the following command:

```
var_dump(openssl_get_cert_locations());
```

which should lead to an output similar to this:

```
array(8) {
  'default_cert_file' =>
  string(32) "/usr/local/etc/openssl/cert.pem"
  'default_cert_file_env' =>
  string(13) "SSL_CERT_FILE"
  'default_cert_dir' =>
  string(29) "/usr/local/etc/openssl/certs"
  'default_cert_dir_env' =>
  string(12) "SSL_CERT_DIR"
  'default_private_dir' =>
  string(31) "/usr/local/etc/openssl/private"
  'default_default_cert_area' =>
  string(23) "/usr/local/etc/openssl"
  'ini_cafile' =>
  string(0) ""
  'ini_capath' =>
  string(0) ""
}
```

Now check if the file given in the `default_cert_file` field actually exists. Create a backup of the file, download the current CA bundle from <https://curl.haxx.se/ca/cacert.pem> and put it where `default_cert_file` points to.

If the problem still occurs, another possible solution is to configure the `curl.cainfo` setting in your `php.ini`:

```
[curl]
curl.cainfo = /absolute/path/to/cacert.pem
```

Migration

3.0 to 3.1

Kreait\Firebase\Factory::withCredentials() has been deprecated

```
# Before
use Kreait\Firebase\Factory;

$firebase = (new Factory)
    ->withCredentials(__DIR__.'/google-service-account.json');

# After
use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;

$serviceAccount = ServiceAccount::fromJsonFile(__DIR__.'/google-service-account.json
    ->);
```

```
$firebase = (new Firebase\Factory)
    ->withServiceAccount($serviceAccount);
```

2.x to 3.x

Database secret authentication

As Database Secret based authentication has been deprecated by Firebase, it has been removed from this library. Use Service Account based authentication instead.

Firebase Factory

Previously, it was possible to create a new Firebase instance with a convenience class in the root namespace. This class has been removed, and `Kreait\Firebase\Factory` is used instead:

```
# Before
$firebase = \Firebase::fromServiceAccount('/path/to/google-service-account.json');

# After
use Kreait\Firebase\Factory;

$firebase = (new Factory())
    ->withCredentials('/path/to/google-service-account.json')
    ->create();
```

Changed namespace

All classes have been moved from the `Firebase` root namespace to `Kreait\Firebase` to avoid conflicts with official Firebase PHP libraries using this namespace.