# DEEZER Native SDK User Guide

# Contents

# 1 Main features and changes

Stay tuned for new releases by subscribing to the Native SDK's release feed.

**Network stack**
The SDK provides a set of network utilities. These utilities are used internally by the SDK but can be used by your application as well. Please read the corresponding Network Stack section.

**Enhanced player**
The *DEEZER Player* of the Native SDK can play contents such as:

- songs (individually)
- albums
- playlists
- radios (aka mixes)

Please refer to the section dedicated to the DEEZER Player for more details.

The *DEEZER Player* supports mixes generated from a playlist, an artist, an album or a user. The latter feature is also known as *Flow*.

**Offline mode and smart cache**
The DEEZER Native SDK provides an offline mode which allows your application to download and store albums and playlists locally. The content marked as *synchronized offline* can then be played without a network connection.

The SDK also provides a smart cache mechanism keeping last played songs on the file system up to the maximum size set by the application. Please read the corresponding Offline Mode section.

**Supported platforms**
List of supported platforms:

| | i386 | x86_64 | ARM |
|---|---|---|---|
| **Mac OS X** (1) | | X | |
| **Windows** (2) | X | X | |
| **Linux** (3) | X | X | X |

(1) Mac OS X 10.8 or upper. (2) Windows7 or upper. (3) Valitaded with Fedora23 and Raspbian Jessie.

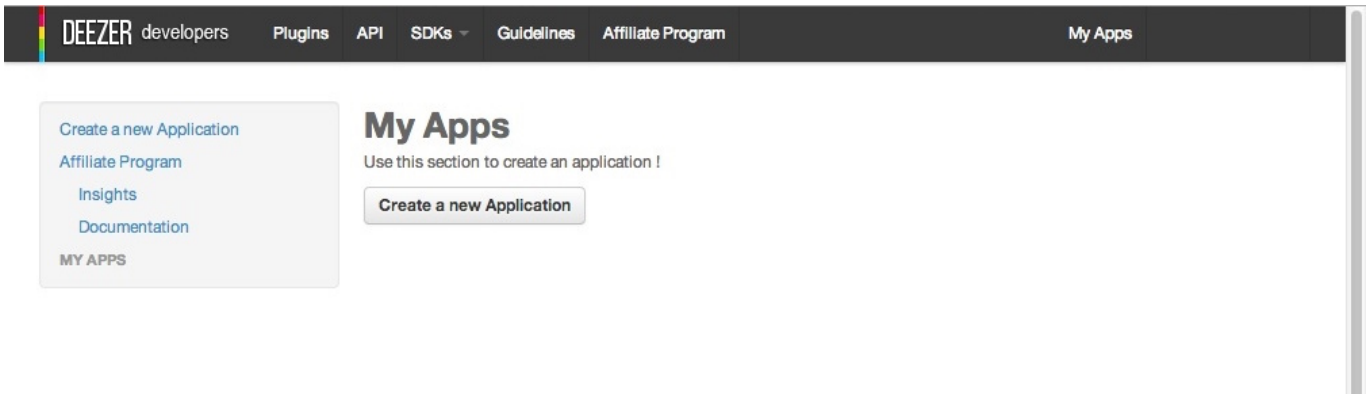
# 2 Quick Start

This section is a step by step guide to get you up and running with the SDK initialisation.

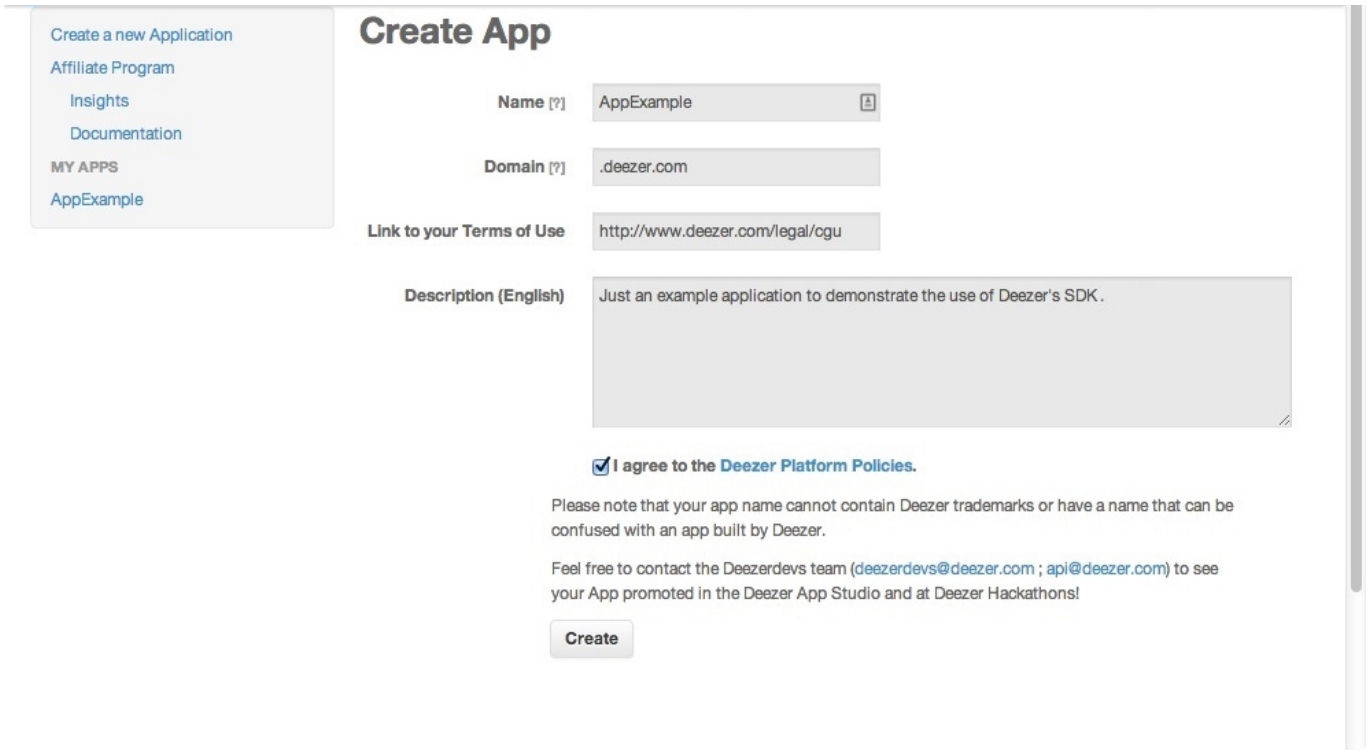
## 2.1 Registering your application with DEEZER

The first step is to register your application with DEEZER. Registering on Deezer will get you an "Application ID" *app_id* and an "Application Secret" *app_secret* which are required to initailize the SDK.

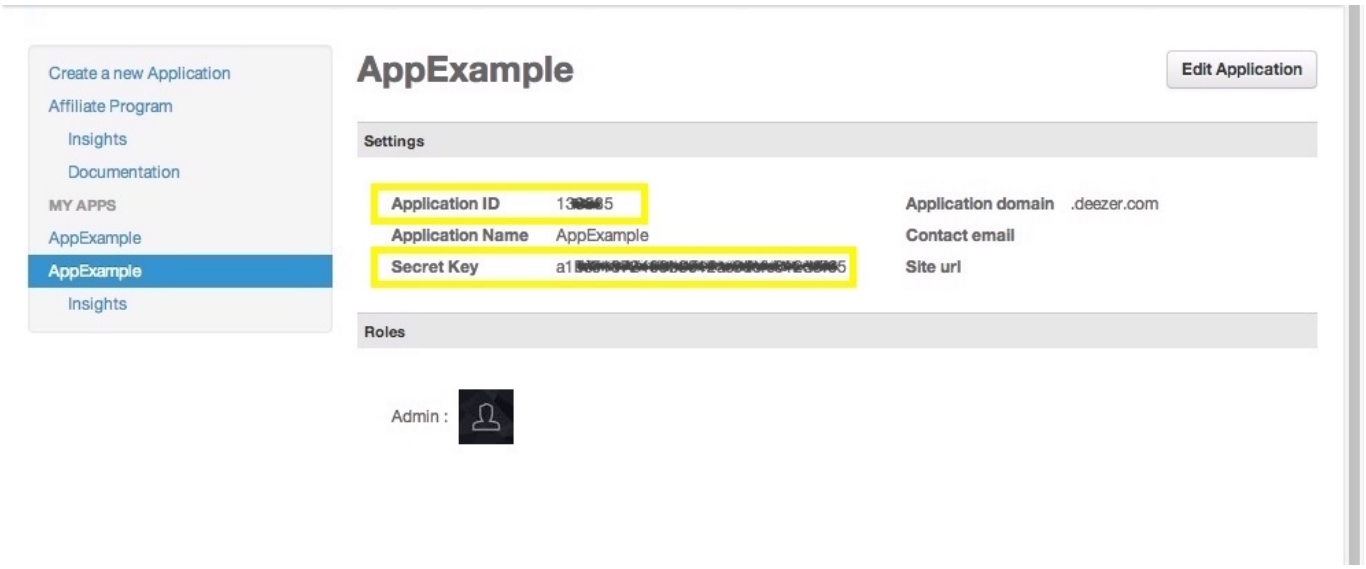
To register your application, you will need a DEEZER user account. Please go to http://developers.deezer.com/, log in with your user account or create one if needed. Then, click on the *My Apps* button next to your login, in the top right-hand corner right of the tab bar.




A screen looking like the one just above will be presented to you. If you already have created your application, click on its name in the left sidebar (it will appear under *MY APPS*) and configure it as described later in this section. Otherwise, click on the button *Create a new Application* to create one.




In order to create your application complete the required fields, tick the *"I agree to the Deezer Platform Policies"* after having read the said document and finally click on *Create* to validate the form.




Your DEEZER application has now been created. Note that it appears under the *MY APPS* section in the left sidebar.

Your application is now properly registered with DEEZER and you can proceed to configure your project.

## 2.2 Configuring your project

To configure your project, you will have to first download unzip the SDK archive. Inside you should find:

- On Mac OS X (10.8 or upper): the framework
- On Windows (Windows7 or upper): the *.dll* files
- On Linux: the *.so* files
- An `include` folder containing the header files

Here is the DEEZER Native SDK file tree:

```
├── Docs
│   ├── Doc.html
│   └── html
├── GettingStarted.pdf
├── Licenses
│   └── ...
└── SDK
    ├── Bins
    │   └── Platforms
    │       ├── Linux
    │       │   ├── arm
    │       │   ├── i386
    │       │   └── x86_64
    │       ├── MacOSX
    │       │   └── libdeezer.framework
    │       └── Windows
    │           ├── DLLs
    │           └── Libs
    └── Include
```

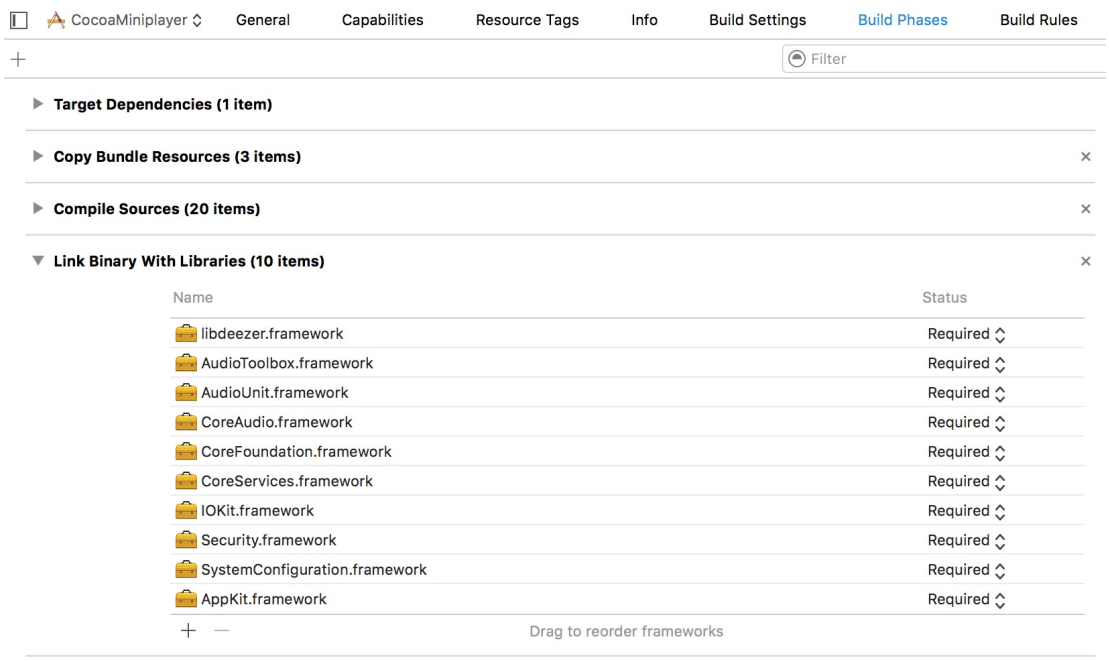
### 2.2.1 External dependencies

#### On Mac OS X

Add the following:

- *AudioToolbox.framework*

- *AudioUnit.framework*
- *CoreAudio.framework*
- *CoreFoundation.framework*
- *CoreServices.framework*
- *IOKit.framework*
- *Security.framework*
- *SystemConfiguration.framework*




#### On Linux

Install the following external package:

- *pulse*

## 2.3 Sample code

Some sample codes are available on the DEEZER GitHub project: https://github.com/deezer/native-sdk-samples

## 2.4 Initializing the SDK

Initializing a `dz_connect` class by calling `dz_connect_new()` **is the very first thing you should do in order to use the DEEZER Native SDK**. To do this, initialize a `dz_connect_configuration` structure with the *app_id* that you got from your DEEZER application page on your DEEZER developer account page. Other members of the `dz_connect_configuration` structure must be set:

**`const char* app_id`**
: The DEEZER application ID being used.

**const char* product_id**
Product ID of the application being used. In a compact form, only ascii characters and . (point character) are allowed.

**const char* product_build_id**
Build product ID which will be used by the Native SDK.

**const char* user_profile_path**
User profile path of application being used.

**dz_connect_onevent_cb connect_event_cb**
(Optional) Connect event callback of application being used.

**const char* anonymous_blob**
(Optional) Use to allow discovery.

**dz_connect_crash_reporting_delegate app_has_crashed_delegate**
(Optional) Delegate used to let the Native SDK know if the application has previously crashed. If this delegate is not set, an internal crash reporter will be used by the Native SDK.

Then you should activate the Native SDK connect instance by calling `dz_connect_activate()` function, this will start the *DEEZER Connect* engine.

# 3 Session Management

To start the login process, you must ask the user about permissions he/she allows you for the application. A list of possible permissions can be found here: Permissions details.

The login process is based on OAuth authentication, which can be implemented through a webview (more details can be found here: OAuth details).

The login needs the *app_id* and *app_secret* that were given to you when registering your app as described in the Quick Start section. Once logged-in, you'll get an *access_token* which you can store so you don't have to go through that process at each start-up.

Furthermore, this token is required to access our API's protected information (the full API list is available here: DEEZER API Explorer).

This can be done by calling `dz_connect_set_access_token()`.

# 4 Network stack

The network stack provided by the DEEZER Native SDK is built to consume Deezer APIs however it can also be used for other purposes. This stack is classical in its architecture, looking a lot like other popular networking frameworks around the native community.

## 4.1 Network Request

All the network requests issued by the DEEZER Native SDK will automatically specify the user *access_token* needed to retrieve user account information. The only essential information you must provide is the *URL* and the *command_type* (GET, POST, DELETE) (the list of all available commands is available here: POST Actions and here DELETE Actions). You have to call `dz_api_request_new()` in order to create a network request. Everything else is optional and depends on the use case.

Then just call `dz_api_request_processing_async()` function to launch the request.

The request response will be returned by the server in the JSON format.

If a JSON stream parser (which must implement the `dz_stream_parser_class` interface) is provided when calling `dz_api_request_processing_async()` the `tokener_parse()` function will be called for each HTTP chunk received. If no parser is provided the raw JSON will be returned only at the end when calling back the `dz_api_request_done_cb()`.

The JSON stream parser implementation is optional.

### 4.2 The result callback `dz_api_request_done_cb()`

If you expect the server to send data back to you, you should provide a `dz_api_request_done_cb()` callback. It will be called when the request finishes (with or without error). If the request completes without any errors and its response contains a body, you will be sent a `dz_stream_object` instance containing the bytes for the response and the `dz_api_result_t` parameter.

If a JSON stream parser has been provided, the `dz_stream_object` will be the object pointer returned by the last `token er_parse()` call. If no JSON stream parser has been provided, the `dz_stream_object` is the raw JSON as a C-String (`const char*`).

On the other hand, if the request encountered an error or was cancelled your callback will be called with an `dz_api_result_t` set and `NULL` for the data.

## 5 Player

The DEEZER Native SDK player allows you to play songs and radios (aka mixes). The player takes *DEEZER URLs* as parameters.

Currently, the *DEEZER Player* supports *DEEZER URLs* schemes such as:

- *"dzmedia:///track/<song_id>"* to play the song with the specified <song_id>.
- *"dzmedia:///album/<album_id>"* to play the album with the specified <album_id>.
- *"dzmedia:///playlist/<playlist_id>"* to play the playlist with the specified <playlist_id>.
- *"dzradio:///radio-<id>"* to play the radio (aka mix) with the specified <id>.
- *"dzradio:///artist-<id>"* to play the artist radio (aka mix) with the specified <id>.
- *"dzradio:///user-<id>"* to play the user radio (aka mix) with the specifide <id>.

At first you need to create an instance of *DEEZER Player* by calling `dz_player_new()`. This function takes a *DEEZER Connect* instance as parameter. Then you have to activate the player instance, this will start the player engine. You will have to wait that the `DZ_CONNECT_EVENT_USER_LOGIN_OK` event has been received in order to go further. If it is not the case please check the Session Management section.

### 5.1 Playing a song

In order to play a content you will need its *DEEZER URL*. Load the content by calling `dz_player_load()` function, this will put the content in cache before it can be played. Finally you have to call `dz_player_play()` function to trigger the playback of the content.

### 5.2 Playing an album, a playlist or a mix

An album, a playlist or a mix can be played with its *DEEZER URL* just as for playing songs. Load the content you want to play by calling `dz_player_load()` function. Finally you have to call `dz_player_play()` to trigger the playback of the content.

You can go to the next song with the `dz_index_in_queuelist`: `DZ_INDEX_IN_QUEUELIST_NEXT`, otherwise just use `DZ_INDEX_IN_QUEUELIST_CURRENT`.

---

**Playing an album or a playlist**
Before calling `dz_player_play()` function be sure that the `DZ_PLAYER_EVENT_QUEUELIST_LOADED` event has been received. If the content fails to be loaded `DZ_PLAYER_EVENT_QUEUELIST_NO_RIGHT` event will be sent instead.

---

### 5.3 Use repeat and shuffle modes

Shuffle and repeat modes are available when playing albums or playlists. These modes can be enabled by calling `dz_player_set_repeat_mode()` and `dz_player_enable_shuffle_mode()`.

**Playing a radio (aka mix) without login**
It's possible to play a radio (aka mix) without having a user logged in.

## 6 Offline Mode

The DEEZER Native SDK also provides an offline mode.

The offline mode is only available to paying users and allows to play content, such as playlists, albums and songs, without an internet connection.

When the offline mode is available for the logged in user, `DZ_CONNECT_EVENT_USER_OFFLINE_AVAILABLE` event is returned through the *DEEZER Connect* event callback function.

If you are interested in using offline mode, the first thing should do is to register an offline callback function in order to be notified of offline events. This is done by using `dz_offline_eventcb_set()` function.