API Developer Documentation

API Site details

http://api.collinsdictionary.com
An api demo facility can be found here:
https://api.collinsdictionary.com/apidemo/ You will need to provide your unique access key to use this facility. See next section.

Unique keys

Your unique keys will be supplied on request. The key will determine access to specific datasets and the number of calls available.

API details

The CollinsDictionary API is RESTful API. Content may be returned in either JSON or XML format. The following standard wrappers are provided to help you get started with development (for more detail see section below) :-

- Java (the reference implementation)
- Perl
- PHP
- JavaScript
- Objective-C

Returned data formats

The API will as standard return data from all queries in either:

- JSON
- XML

The return format depends on the "Accept" HTTPS header:

- **application/json** => JSON
- **application/xml** => XML
- no accept header => JSON
- any other accept header => 406 status

General format

API methods have this general format:
http://<hostname>/api/v1/...
where:

- http://api.collinsdictionary.com is a CollinsDictionary website. Note that only the https: protocol is supported.
- "api" is a constant, indicating that this is an API request to the host.
- "v1" indicates that this request uses version 1 of the API. (Future versions of the API will introduce "v2", "v3", etc.)
- ... indicates the body of the request. This is detailed below for each method.

API Specification - Methods

Get list of available dictionaries

REST request = /api/v1/dictionaries

INPUT

No parameter

JSON OUTPUT

1. an array of available dictionaries (empty if no dictionaries are found):
    a. dictionaryCode - the code of the dictionary
    b. dictionaryName - the user-friendly name of the dictionary
    c. dictionaryUrl - the URL of the dictionary's browse list.

XML OUTPUT

```
<dictionaries>
<dictionary>
<dictionaryCode>{dictionaryCode}</dictionaryCode>
<dictionaryName>{dictionaryName}</dictionaryName>
<dictionaryUrl>{dictionaryURL}</dictionaryUrl>
</dictionary>
...
</dictionaries>
```

An empty "dictionaries" tag is returned if no dictionaries are found

Get a dictionary

REST request = /api/v1/dictionaries/{dictionaryCode}

INPUT

1. dictionaryCode - the dictionary code

JSON OUTPUT

1. dictionaryCode - the code of the dictionary
2. dictionaryName - the user-friendly name of the dictionary
3. dictionaryUrl - the URL of the dictionary's browse list.

XML OUTPUT

```
<dictionary>
<dictionaryCode>{dictionaryCode}</dictionaryCode>
<dictionaryName>{dictionaryName}</dictionaryName>
<dictionaryUrl>{dictionaryURL}</dictionaryUrl>
</dictionary>
```

An error is returned if the dictionary does not exist.

Search

REST request = /api/v1/dictionaries/{dictionaryCode}/search/?q={searchWord}&pagesize={pageSize}&pageindex={pageIndex}

INPUT

1. dictionaryCode - the dictionary code.
2. searchWord - the word we are searching for.
3. pageSize - the number of results per result page [optional, 10 by default, maximum allowed 100]
4. pageIndex – the index of the result page to return [optional, default = 1 (the first page)]

JSON OUTPUT

1. dictionaryCode
2. resultNumber – the total number of results
3. pageNumber – the index of the last result page
4. currentPageIndex – the index of the current result page
5. results - an array of entries sorted the same way as in the main dictionary (empty if no results are found):
    a. entryId - the id of the entry
    b. entryLabel - the label of the entry (headword)
    c. entryUrl - the direct url to the entry page on the main website

XML OUTPUT

```
<search><dictionaryCode>{dictionaryCode}</dictionaryCode><resultNumber>{resultNumber}</resultNumber><pageNumber>{pageNumber}</pageNumber><currentPageIndex>{currentPageIndex}</currentPageIndex><results><entry><entryId>{entryId}</entryId><entryLabel>{entryLabel}</entryLabel><entryUrl>{entryUrl}</entryUrl></entry> <entry>...</entry></results></search>
```

An error is returned if the dictionary does not exist.
An empty "results" tag is returned if no results are found

Get Did You Mean entries

REST request = /api/v1/dictionaries/{dictionaryCode}/search/didyoumean/?q={searchWord}&entrynumber={entryNumber}

INPUT

1. dictionaryCode - the dictionary code
2. searchWord - the word we are searching for
3. entryNumber - the number of Did you Mean results [optional, 10 by default, maximum allowed 10]

JSON OUTPUT

1. dictionaryCode
2. searchTerm - the word we are searching for
3. suggestions - the list of suggestions proposed by the Did you Mean feature
    a. suggestion - the suggestion

An empty "suggestions" array is returned if no results are found

XML OUTPUT

```
<didyoumean>
   <dictionaryCode>{dictCode}</dictionaryCode>
   <searchTerm>{searchWord}</searchTerm>
   <suggestions>
      <suggestion>{suggestion}</suggestion>
      <suggestion>{suggestion}</suggestion>
      <suggestion>{suggestion}</suggestion>
      ...
   </suggestions>
</didyoumean>
```

An empty "suggestions" array is returned if no results are found

Get the first/best matching entry

REST request = /api/v1/dictionaries/{dictionaryCode}/search/first/?q={searchWord}&format={format}

INPUT

1. dictionaryCode - the dictionary code
2. searchWord - the word we are searching for
3. format - the format of the entry, either "html" or "xml" [optional; default = html]

JSON OUTPUT

1. dictionaryCode
2. format
3. entryContent
4. entryId - the id of the entry
5. entryLabel - the label of the entry (headword)
6. entryUrl - the direct url to this entry on the main website
7. topics - an array containing the topics linked to the entry (if any):
    a. topicId - the id of the topic
    b. topicLabel - the label of the topic
    c. topicUrl - the direct url to the topic page on the main website
    d. topicParentId - the id of the parent topic (if any)

XML OUTPUT

```
<entry><dictionaryCode>{dictionaryCode}</dictionaryCode><format>{forma
t}</format><content>{entryContent}</content><id>{entryId}</id><label>{
entryLabel}</label><url>{entryUrl}</url>><topics><topic><id>{topicId}<
/id><label>{topicLabel}</label><url>{topicUrl}</url><parentId>{topicPa
rentId}</parentId></topic><topic>...</topic></topics></entry>
```

An error is returned if the dictionary does not exist or if no results are found.

Get an entry

REST request = /api/v1/dictionaries/{dictionaryCode}/entries/{entryId}?format={format}

INPUT

1. dictionaryCode - the dictionary code
2. entryId - the id of the entry
3. format - the format of the entry, either "html" or "xml" [optional; default = html]

JSON OUTPUT

1. dictionaryCode
2. format
3. entryContent
4. entryId - the id of the entry
5. entryLabel - the label of the entry (headword)
6. entryUrl - the direct url to this entry on the main website
7. topics - an array containing the topics linked to the entry (if any):
    a. topicId - the id of the topic
    b. topicLabel - the label of the topic
    c. topicUrl - the direct url to the topic page on the main website
    d. topicParentId - the id of the parent topic (if any)

An empty JSON object is returned if no results are found

XML OUTPUT

```
<entry><dictionaryCode>{dictionaryCode}</dictionaryCode><format>{forma
t}</format><entryContent>{entryContent}</entryContent><entryId>{entryI
d}</entryId><entryLabel>{entryLabel}</entryLabel><entryUrl>{entryUrl}<
/entryUrl><topics><topic><id>{topicId}</id><label>{topicLabel}</label>
<url>{topicUrl}</url><parentId>{topicParentId}</parentId></topic><topi
c>...</topic></topics></entry>
```

The entry will be escaped appropriately so that it forms a valid string in the selected return format (JSON or XML).
An error is returned if the dictionary does not exist or if the entry does not exist.

Get pronunciations

REST request = /api/v1/dictionaries/{dictionaryCode}/entries/{entryId}/pronunciations?lang={lang}

INPUT

1. dictionaryCode - the dictionary code
2. entryId - the id of the entry
3. lang [optional; default = all available pronunciation languages]

JSON OUTPUT

1. an array of pronunciations
    a. dictionaryCode
    b. entryId
    c. lang
    d. pronunciationUrl - the url of the mp3 file

An empty JSON object is returned if no results are found

XML OUTPUT

```
<pronunciations><pronunciation><dictionaryCode>{dictionaryCode}</dictionaryCode><entryId>{entryId}</entryId><lang>{lang}</lang><url>{pronunciationUrl}</url></pronunciation><pronunciation>...</pronunciation><pronunciations>
```

An empty "pronunciations" tag is returned if no results are found.
An error is returned if the dictionary does not exist.

Get nearby entries

REST request =
/api/v1/dictionaries/{dictionaryCode}/entries/{entryId}/nearbyentries?entrynumber={entryNumber}

INPUT

1. dictionaryCode - the dictionary code
2. entryId - the id of the entry
3. entryNumber - the number of results preceding/following the given entry [optional, 10 by default, maximum allowed 50]

JSON OUTPUT

1. dictionaryCode
2. entryId - the id of the entry
3. nearbyFollowingEntries - an array of the entries preceding the given entry
    a. entryId - the id of entry
    b. entryLabel - the headword of the entry
    c. entryUrl - the direct url to the entry page on the main website
4. nearbyPrecedingEntries - an array of entries following the given entry
    a. entryId - the id of entry
    b. entryLabel - the headword of the entry
    c. entryUrl - the direct url to the entry page on the main website

An empty JSON object is returned if no results are found

XML OUTPUT

```
<entry>
<dictionaryCode>{dictionaryCode}</dictionaryCode>
<entryId>{entryId}</entryId>
<nearbyFollowingEntries>
<nearbyEntry>
<entryId>{entryId}</entryId>
<entryLabel>{entryLabel}</entryLabel>
<entryUrl>{entryUrl}</entryUrl>
</nearbyEntry>
<nearbyEntry>
...
</nearbyEntry>
</nearbyFollowingEntries>
<nearbyPrecedingEntries>
<nearbyEntry>
<entryId>{entryId}</entryId>
<entryLabel>{entryLabel}</entryLabel>
<entryUrl>{entryUrl}</entryUrl>
```

```
</nearbyEntry>
<nearbyEntry>
...
</nearbyEntry>
</nearbyPrecedingEntries>
<entry>
```

If no results are found, empty "nearbyPrecedingEntries" / "nearbyFollowingEntries" tags are returned.
An error is returned if the dictionary does not exist or the entry does not exist.

Online documentation

Documentation of the methods will be available at:

- https://api.collinsdictionary.com/api/v1/documentation/html – HTML format.
- https://api.collinsdictionary.com/api/v1/documentation/wadl – XML (WADL) format

Manage Errors/Exceptions

If the call of a method raises an exception, the returned HTTPS status code is 5xx and the response will be formatted this way:

JSON OUTPUT

1. errorCode
2. errorMessage

XML OUTPUT

```
<error>
<code>{errorCode}</code>
<message>{errorMessage}</message>
</error>
```

Full details of error codes will be provided in the online documentation.

API Client Libraries

All libraries are "proofs of concept" that illustrate how to develop a simple interface between your application and the API server. They are provided with sample test code that accesses several features of a PitchLeads site using this interface.

For security reasons, it is recommended that the API and these libraries are only used in server-side applications in a client-server architecture. They should not be used in client-side applications.

- Java Library
- C# Library
- Objective-C
- Perl Library
- PHP Library
- Ruby Library
- Python Library
- JavaScript Library

All libraries can be found here - http://dps.api-lib.idm.fr/libraries.html#js

Java Library

You can integrate this library into many Java applications that perform HTTPS requests.

In the sample, using the library simplifies the request process to the API server and directly returns the result object in string form.

Note: the sample code requires the HttpComponents Client (http://hc.apache.org/httpcomponents-client) and JSON library (http://www.json.org/java/index.html)

To use the library, you need to initialize a request handler and an API object:

```
DefaultHttpClient httpClient = new DefaultHttpClient(new
ThreadSafeClientConnManager()); SkPublishAPI api = new
SkPublishAPI(baseUrl + "/api/v1", accessKey, httpClient);
api.setRequestHandler(new SkPublishAPI.RequestHandler() { public void
prepareGetRequest(HttpGet request) { System.out.println(request.getURI());
request.setHeader("Accept", "application/json"); } });
```

Then, simply use the "api" object:

```
try { System.out.println("*** Dictionaries"); JSONArray dictionaries = new
JSONArray(api.getDictionaries()); System.out.println(dictionaries); JSONObject
dict = dictionaries.getJSONObject(0); System.out.println(dict); String dictCode
= dict.getString("dictionaryCode"); System.out.println("*** Search");
System.out.println("*** Result list"); JSONObject results = new
JSONObject(api.search(dictCode, "ca", 1, 1)); System.out.println(results);
System.out.println("*** Spell checking"); JSONObject spellResults = new
JSONObject(api.didYouMean(dictCode, "dorg", 3));
System.out.println(spellResults); System.out.println("*** Best matching");
JSONObject bestMatch = new JSONObject(api.searchFirst(dictCode, "ca",
"html")); System.out.println(bestMatch); System.out.println("*** Nearby
Entries"); JSONObject nearbyEntries = new
JSONObject(api.getNearbyEntries(dictCode, bestMatch.getString("entryId"),
3)); System.out.println(nearbyEntries); } catch (Throwable t) {
t.printStackTrace(); }
```

Web page functionality

The following JavaScript libraries are also available:

- Double click – dictionary definition
- Hover over (on tagged words) – dictionary definition
- Right click

Widgets

Collins Dictionary provides the following widget:

- Branded widget that can be easily included in your page and opens a Collinsdictionary.com definition page - http://www.collinsdictionary.com/tools-and-widgets