Configuration

After installing the Phrase Client, learn how to set it up for your project.

Create a configuration file

To create a configuration file, you'll need to install the Phrase Client if you haven't already. You can then create your **`.phraseapp.yml`** configuration file by running

```
$ phraseapp init
```

You will need to specify:

- Your access token (which you can create from the Translation Center in your profile settings)
- Your Phrase project ID
- A locale file format
- The location of your locale files inside your project's codebase

This command will generate a basic configuration file in the current working directory.

If you decide to create a configuration file from scratch, you should place it in one of these locations:

- The directory from which you called the CLI client
- The current user's home directory (**`$HOME`** on UNIX OSs, or **`$HomePath`** on Windows)
- The path specified in the **`PHRASEAPP_CONFIG`** environment variable

Modify your configuration file

You can use the **`.phraseapp.yml`** file to store command line arguments you don't want to specify every time you run the client, such as a project ID.

The following arguments are supported:

project_id

The ID of your Phrase project.

file_format

The file format used when one is not specified explicitly within source and target entries. For available file formats and their API extensions, refer to the format guide.

per_page

The number of items returned in paginated responses.

defaults

The default parameters for specific API actions, specified as a list of keys and values.

This is long version of the command to list all keys for a project:

```
$ phraseapp keys list \

--access-token 3d7e6598d955bfcabaf1b9459df5692ac4c28a17793 \

--sort updated_at --order desc 5c05692a2a995c0c45c0c3cbfcab1
```

You can move some of these arguments to your **`.phraseapp.yml`** configuration file:

```
phraseapp:

  access_token: 3d7e6598d955bfcabaf1b9459df5692ac4c28a17793

  project_id: 5c05692a2a995c0c45c0c3cbfcab1

  file_format: yml

 defaults:

    keys/list:

      sort: "updated_at"

      order: "desc"

...
```

After doing so, you can shorten the command line:

```
$ phraseapp keys list
```

We recommend moving arguments to the configuration file to make using the command-line client easier and more consistent (e.g., across team members).

Push and pull

Use the **`push`** command to import locale files to Phrase:

```
$ phraseapp push
```

The **`push`** command uploads files found in your local project directories. Use the **`.phraseapp.yml`** file to specify the files that should be uploaded, and to set any additional parameters you need.

Use the **`pull`** command to download locale files from Phrase:

```
$ phraseapp pull
```

This command works similarly to the **`push`** command, with the exception that you cannot use globbing (see below). When using placeholders, we recommend using **`<locale_name>`** whenever possible. Alternatively, you can use the **`<tag>`** placeholder to download keys into separate files based on their tags.

Parameters

You can configure the **`push`** command within the `.phraseapp.yml` configuration file. You can use the same options that are available for the **`uploads#create`** API endpoint. Similarly, the **`pull`** command supports the same options that are available for the **`locales#download`** API endpoint.

You can specify options like this:

Push

```
push:

    sources:

    - file: ./locales/en.json

       params:

        update_translations: true
```

Pull

```
pull:

   targets:

   - file: "./locales/example.yml"

       params:

       include_unverified_translations: true
```

Format options

Depending on the file format in use, you can also add a variety of format options to the parameter section. Refer to the list of supported platforms and formats to find your file format and the list of options supported for it. Format options can be available for upload, download or both,

You can specify format options in the parameter section like this:

```
params:

 format_options:

   convert_placeholder: true
```

Placeholders and globbing

You can use the following placeholders and globbing operators in the paths within your file entries:

<locale_name>

The locale name is the unique name of your locale in Phrase.

<locale_code>

The locale code is the RFC 5646-compliant locale identifier. Note that the locale code does not have to be unique, so you can have multiple locales with different names but the same code.

<tag>

Use tags to group keys in Phrase. This is especially useful when you want to keep the original structure of your files.

Globbing

* and ** are globbing operators. A single asterisk * skips any folder in a path. The double asterisk works like the standard globbing operator and matches any character. ** therefore stands for recursive, non-exhaustive matching.

```
# a file pattern

./abc/**/*.yml



# with a few files on your system

./abc/defg/en.yml

./abc/es.yml

./fr.yml
```

```
# selects
./abc/defg/en.yml
./abc/es.yml
```

When using the `pull` command to download files, you must provide an explicit file pattern like **`./abc/defg/<locale_name>.yml`** instead of **`./**/*.yml`**.

Waiting for uploads

All uploads are processed asynchronously. To wait for uploads, pass the **`--wait`** flag. This tells **`push`** to wait for each file upload and returns whether it failed or succeeded.

Common uses

```
phraseapp:
  access_token: 3d7e6598d955bfcabaf1b9459df5692ac4c28a17793
  project_id: 5c05692a2a995c0c45c0c3cbfcab1
  file_format: nested_json

  push:
    sources:
    - file: ./locales/en.json
      params:
        update_translations: false
        locale_id: YOUR_LOCALE_ID # the locale must exist remotely
```

This example will upload the file **`en.json`** in the **`./locales/`** directory to Phrase with the given **`locale_id`** (note that the locale must already exist in Phrase). In this example, **`update_translations`** is set to **`false`**, so only new keys and translations will be imported into Phrase. If it were set to **`true`**, the client would also import local changes to existing translations, overwriting any content already present in Phrase.

Here is an example for Ruby on Rails YAML files:

```
phraseapp:

  push:

    sources:

    - file: ./config/locales/*.yml

      params:

        update_translations: true

        file_format: yml
```

These settings will upload all files ending with **`.yml`** located in **`./config/locales/`** to Phrase. Note that Ruby on Rails YAML files contain locale information, so there is no need to specify the locale explicitly. In this example, **`update_translations`** is set to **`true`**, so all changes to translations will be imported into Phrase. Any data already present in Phrase will be overwritten.

The following example for iOS strings files matches all files named **`Localizable.strings`** in your .lproj folders:

```
phraseapp:

  push:

    sources:

    - file: "./<locale_code>.lproj/Localizable.strings"

      params:

        convert_emoji: true

        file_format: strings
```

The **`<locale_code>`** is everything that matches after / and before .lproj. It is used to create and identify locales in Phrase.

In this example, we omitted the **`update_translations`** parameter, which has the same effect as setting it to **`false`**.

Configuration examples for popular frameworks

Rails

```yaml
phraseapp:
  access_token: "3d7e6598d955bfcab104c45c037af1b9459df5692ac4c28a17793"
  project_id: "5c05692a2a995c0c45c0c3cbfcab1"
  file_format: "yml"
  push:
    sources:
    - file: "./config/locales/*.yml"
  pull:
    targets:
    - file: "./config/locales/<locale_name>.yml"
```

iOS strings

```yaml
phraseapp:
  access_token: "3d7e6598d955bfcab09f232a99c37af1b9459df5692ac4c28a17793"
  project_id: "5c05692a2a995c0c45c0c3cbfcab1"
  file_format: "strings"
  push:
    sources:
    - file: "./<locale_code>.lproj/Localizable.strings"
      params:
        convert_emoji: true
  pull:
    targets:
    - file: "./<locale_code>.lproj/Localizable.strings"
      params:
        convert_emoji: true
    - file: "./<locale_code>.lproj/Localizable.stringsdict"
      params:
        # access_token or file_format can be overwritten
        file_format: "stringsdict"
```

Android XML

```
phraseapp:
  access_token: "3d7e6598d955bfcab109f232a99c37af1b9459df5692ac4c28a17793"
  project_id: "5c05692a2a995c0c45c0c3cbfcab1"
  file_format: "xml"
  push:
    sources:
    - file: "./res/values-<locale_code>/strings.xml"
  pull:
    targets:
    - file: "./res/values-<locale_code>/strings.xml"
```

We recommend modifying the configuration file to suit your needs. Then, check it into your source control or version control system.