GOOGLE APP ENGINE

CHAPTER 10

PHP: TOOLS

Información extraida de la documentación oficial de GOOGLE APP ENGINE, recopilada en PDF para su mejor distribucción. A menos que se indique lo contrario, el contenido de esta página tiene la Licencia de Creative Commons Atribución 3.0, y las muestras de código tienen la Licencia Apache 2.0. Para obtener más información, consulta las Políticas de Google App Engine.

The PHP Development Server

The App Engine PHP SDK includes a web server application you can run on your computer that simulates your application running in the App Engine PHP runtime environment. The simulated environment enforces some sandbox restrictions, such as restricted system functions and PHP module imports, but not others, like request time-outs or quotas. The server also simulates the services by performing their tasks locally.

Running your own PHP interpreter in the Development Server

The Mac OS X and Windows PHP SDKs include binaries for the PHP 5.4 runtime so there is usually no need to install PHP separately for the purposes of developing with App Engine. However, the local development environment that is installed with the SDK is missing the enabled extensions mcrypt and gd, so if you want to simulate their use in your local development environment, you may wish to install your own PHP php-cgi interpreter.

When running the Development Server from the command line, you can specify that your custom version of the PHP execuable be used with the `--php_executable` flag, eg.

```
google_appengine/dev_appserver.py --php_executable_path=/opt/local/bin/php-cgi54
path/to/your/app
```

Building a custom interpreter on Mac OS X using MacPorts

On Mac OS X, you can install a custom PHP interpeter with gd and mycrypt using MacPorts. If you don't already have MacPorts installed, you should follow the installation instructions (including the installation of Xcode).

Once MacPorts is installed, you can install PHP 5.4 with the following command. (Note: This example also demonstrates how to install mcrypt along with PHP 5.4):

```
sudo /opt/local/bin/port install php54-cgi php54-APC php54-calendar \

php54-exif php54-gd php54-mysql php54-oauth php54-openssl php54-soap \

php54-xdebug php54-xsl php54-mcrypt
```

After installation, the `php-cgi` (the binary used by the PHP SDK) can be found at `/opt/local/bin/php-cgi54`.

Running the development web server

Once you have a directory for your application and an app.yaml configuration file, you can start the development web server with the `dev_appserver.py` command:

```
google_appengine/dev_appserver.py path-to-your-app
```

The web server listens on port 8080 by default. You can visit the application at this URL: http://localhost:8080/.

To change which port the web server uses, use the `--port` option:

```
google_appengine/dev_appserver.py --port=9999 path-to-your-app
```

To stop the web server: with Mac OS X or Unix, press Control-C or with Windows, press Control-Break in your command prompt window.

Application IDs in the development web server

If you need to access your App ID, for example to spoof an email address, use the`AppIdentityService::getApplicationId` function. To get the hostname of the running app, use the`AppIdentityService::getDefaultVersionHostname` function.

Warning: Do not get the App ID from the environment variable. The development server simulates the production App Engine service. One way in which it does this is to prepend a string (`dev~`) to the`APPLICATION_ID` environment variable. Google recommends always getting the application ID using`AppIdentityService::getDefaultVersionHostname`, as described above.

Storing data in the development web server

Google App Engine for PHP supports reading and writing to Google Cloud Storage via PHP's streams API. A developer can read to and write from an object in Google cloud storage by specifying it as a URI to any PHP function that supports PHPs Streams implementation such as `fopen()`, `fwrite()` or `get_file_contents()`.

In the Development Server, when a Google Cloud Storage URI is specified we emulate this functionality by reading and writing to temporary files on the user's local filesystem. These files are preserved between requests, allowing you to test the functionality on your local development environment before deploying your code to App Engine.

In the PHP development server streaming calls
like `fopen()`, `file_get_contents()` on `'gs://'` urls are mocked by reading and writing to the local filesystem.

Browsing the local Datastore

To browse your local Datastore using the development web server:

1. Start the development server as described previously.
2. Go to the Development Console.
3. Click Datastore Viewer in the left navigation pane to view your local Datastore contents.

The Users service in the development web server

App Engine provides a Users Service to simplify authentication and authorization for your application. The development web server simulates the behavior of Google Accounts with its own sign-in and sign-out pages. While running under the development web server, the `createLoginURL` and `createLogoutURL` functions return URLs for `/_ah/login` and `/_ah/logout` on the local server.

Using Mail

The development web server can send email for calls to the App Engine mail service. To enable email support, the web server must be given options that specify a mail server to use. The web server can use an SMTP server, or it can use a local installation of Sendmail.

To enable mail support with an SMTP server, use the `--smtp_host`, `--smtp_port`, `--smtp_user` and `--smtp_password` options with the appropriate values.

```
dev_appserver.py --smtp_host=smtp.example.com --smtp_port=25 \
    --smtp_user=ajohnson --smtp_password=k1tt3ns myapp
```

To enable mail support with Sendmail, use the `--enable_sendmail` option with a value of `yes`. The web server will use the `sendmail` command to send email messages with your installation's default configuration.

```
dev_appserver.py --enable_sendmail myapp
```

If mail is not enabled with either SMTP or Sendmail, then attempts to send email from the application will do nothing, and appear successful in the application.

Using URL Fetch

When your application uses the URL fetch API to make an HTTP request, the development web server makes the request directly from your computer. The behavior may differ from when your application runs on App Engine if you use a proxy server for accessing websites.

Local debugging with XDebug

If you have a debugger that is compatible with the XDebug debugger, and you have the xdebug module installed, it is possible to use XDebug with the Development Server to debug your local application. To enable XDebug on the Development Server on Linux or Mac OS X you need to perform two steps:

1. Export the XDEBUG_CONFIG environment variable with an idekey for your IDE to connect to

```
export XDEBUG_CONFIG="idekey=netbeans-xdebug remote_host=localhost"
```

2. Invoke the Development Server with `--php_remote_debugging=yes`

The Development Console

The development web server includes a console web application. With the console, you can browse the local datastore, and interact with the application by submitting Python code to a web form.

To access the console, visit the URL `http://localhost:8000` on your server.

Command-line arguments

The `dev_appserver.py` command supports the following command-line arguments:

`--auto_id_policy=...`

Deprecated. How the local datastore assigns automatic IDs. Options are `sequential` or `scattered`. The default is `scattered`.

`--admin_host=ADMIN_HOST`

Host name to which the local Development Console should bind (default: localhost).

`--admin_port=ADMIN_PORT`

Port to which the local Development Console should bind (default: 8000).

`--clear_datastore=yes|no`

Clears the datastore data and history files before starting the web server. The default is `no`.

`--datastore_path=...`

The path to use for the local datastore data file. The server creates this file if it does not exist.

`--enable_sendmail`

Uses the local computer's Sendmail installation for sending email messages.

--help

Prints a helpful message then quits.

--host=...

The host address to use for the server. You may need to set this to be able to access the development server from another computer on your network. An address of `0.0.0.0` allows both localhost access and hostname access. Default is `localhost`.

--log_level=...

The lowest logging level at which logging messages will be written to the console; messages of the specified logging level or higher will be output. Possible values are `debug`, `info`, `warning`, `error`, and`critical`.

--port=...

The port number to use for the server. Default is `8080`. If multiple servers are launched, they will be assigned subsequent ports (e.g. `8081`, `8082`, etc).

--logs_path=LOGS_FILE

> By default, development server logs are stored in memory only. This option turns on disk storage of logs at the location specified by LOGS_FILE, making the logs available across server restarts. For example,
>
> ```
> dev_appserver.py --logs_path=/home/logs/boglogs bog
> ```

--require_indexes=yes|no

Disables automatic generation of entries in the `index.yaml` file. Instead, when the application makes a query that requires that its index be defined in the file and the index definition is not found, an exception will be raised, similar to what would happen when running on App Engine. The default value is `no`.

--smtp_host=...

The hostname of the SMTP server to use for sending email messages.

--smtp_port=...

The port number of the SMTP server to use for sending email messages.

--smtp_user=...

The username to use with the SMTP server for sending email messages.

--smtp_password=...

The password to use with the SMTP server for sending email messages.

--storage_path=...

Path at which all local files (such as the Datastore, Blobstore files, Google Cloud Storage Files, logs, etc) will be stored, unless overridden by `--datastore_path`, `--blobstore_path`, `--logs_path`, etc.

--php_executable_path=...

Path to the php-cgi binary

--php_remote_debugging=yes|no

Set to yes to enable remote debugging with XDebug

Uploading, Downloading, and Managing a PHP App

The App Engine PHP SDK includes a command for interacting with App Engine named `appcfg.py`. You can use this command to upload new versions of the code, configuration and static files for your app to App Engine. You can also use the command to download log data.

Note: If you created your project using the Google Developers Console, your project has a title and an ID. In the instructions that follow, the *project* title and ID can be used wherever an *application* title and ID are mentioned. They are the same thing.

Uploading the app

To upload application files, run the `appcfg.py` command with the `update` action and the name of your application's root directory. The root directory should contain the `app.yaml` file for the application.

```
appcfg.py update myapp/
```

`appcfg.py` gets the application ID from the `app.yaml` file, and prompts you for the email address and password of your Google account. After successfully signing in with your account, `appcfg.py` stores a "cookie" so that it does not need to prompt for a password on subsequent attempts.

You can specify the email address on the command line using the `--email` option. You cannot specify the password as a command line option.

```
appcfg.py --email=Albert.Johnson@example.com update myapp/
```

Only application owners and the developer who uploaded the code can download it. If anyone else attempts to download the app, they'll see an error message like the following:

```
Fetching file list...

Email: user@example.com

Password for user@example.com:

Error 403: --- begin server output ---

You do not have permission to download this app version.
```

```
--- end server output ---
```

Note: Currently, App Engine has a limit of 10 deployed versions per application. If you try to deploy an 11th version you'll get an error: "Too Many Versions (403)." You can use the Admin Console to delete an older version and then upload your latest code.

Updating Task Queue configuration

You can update just the configuration for an app's task queues without uploading the full application. To upload the `queue.yaml` file, use the `appcfg.py update_queues` command:

```
appcfg.py update_queues myapp/
```

Updating the DoS protection configuration

You can update just the configuration for the DoS Protection for an app without uploading the full application. To upload the `dos.yaml` file, use the `appcfg.py update_dos` command:

```
appcfg.py update_dos myapp/
```

Managing scheduled tasks

App Engine supports scheduled tasks (known as cron jobs). You specify these in a file called `cron.yaml`, and upload them using the `appcfg.py update_cron` command:

```
appcfg.py update_cron myapp/
```

`appcfg update` will also upload cron job specifications if the file exists. For more on cron jobs, see the Cron Jobs documentation.

`appcfg cron_info` displays a summary of the scheduled task configuration, and the expected times of the next few runs.

Downloading source code

You can download an application's source code by running appcfg.py with the `download_app` action in the PHP SDK command-line tool:

```
appcfg.py download_app -A <your_app_id> -V <your_app_version> <output-dir>
```

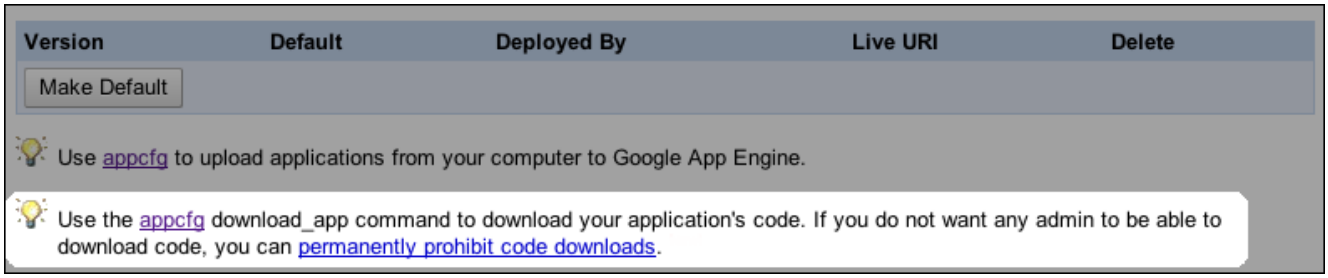
Output like the following results if the command is successful:

```
Getting file list...

Email: <admin-id>@example.com

Password for <admin-id>@example.com:

Fetching files...

Getting files...

[1/5] request.php

[2/5] login.php

[3/5] static/screen.css

[4/5] static/print.css

[5/5] images/bird.png
```

Only the developer who uploaded the code and the application owner(s) can download it. If anyone other than these parties attempts to download the app, they'll see an error message like the following:

```
Fetching file list...

Email: user@example.com

Password for user@example.com:

Error 403: --- begin server output ---

You do not have permission to download this app version.

--- end server output ---
```

If you are listed as an owner or a developer for an application, you can permanently disable code downloads from the Versions screen in your development web server. On that page, click permanently prohibit code downloads:



Warning: This action is irreversible. After you prohibit code download, there is no way to re-enable this feature.

Downloading logs

App Engine maintains a log of messages that your application emits using the `syslog()` module from the PHP standard library, as well as other messages printed to the standard error stream. App Engine also records each request in the log. Each log level has a fixed buffer size that controls the amount of log information you can access. Normally, you use logging features more at lower log levels; thus, the time window is smaller for log events at these levels. You can browse your app's logs of the last 90 days from the "Logs" section of the Admin Console.

If you wish to perform more detailed analysis of your application's logs, you can download the log data to a file on your computer. To download logs to a file named `mylogs.txt`, use the following command:

```
appcfg.py request_logs myapp/ mylogs.txt
```

By default, the command downloads log messages from the current calendar day (since midnight Pacific Time) with a log level of INFO or higher (omitting DEBUG level messages). The command overwrites the local log file. You can adjust the number of days, the minimum log level, and whether to overwrite or append to the local log file using command-line options. See below for more information on these options.

You can limit the log messages that are downloaded to just those emitted during request on a given domain name using the `--vhost=...` option. You can use this to download the logs for your live app using a Google Apps domain or `http://your_app_id.appspot.com`, excluding log messages emitted by versions you are testing on URLs such as `http://2.latest.your_app_id.appspot.com`. Or you can use it to download just the log messages for a given test domain.

There are many options for the `request_logs` command. You can view them all by typing:

```
appcfg.py help request_logs
```

Using an HTTP proxy

If you are running `appcfg.py` behind an HTTP proxy, you must tell `appcfg.py` the name of the proxy. To set an HTTP proxy for `appcfg.py`, set the `http_proxy` and `https_proxy` environment variables.

Using Windows (in Command Prompt):

```
set HTTP_PROXY=http://cache.mycompany.com:3128

set HTTPS_PROXY=http://cache.mycompany.com:3128

appcfg.py update myapp
```

Using the command line in Mac OS X (in Terminal) or Linux:

```
export http_proxy="http://cache.mycompany.com:3128"

appcfg.py update myapp
```

Password-less Login with OAuth2

If you don't want to enter your login credentials, you can use an OAuth2 token instead. This token gives access to App Engine, but not to other parts of your Google account; if your Google account uses two-factor authentication, you'll find this especially convenient. You can store this token to *permanently* log in on this machine.

To set this up, use the `--oauth2` option. (Or, if your browser is on a different machine, perhaps because you're shelled into it, pass `--noauth_local_webserver --oauth2`.) This will store your `OAuth2` credentials in your home directory in a file called `.appcfg_oauth2_tokens`. If you don't want to permanently log in by storing your OAuth2 token on disk, also use the `--no_cookies` option.

```
appcfg.py --oauth2 update myapp/
```

A page will appear in your web browser prompting you for authentication. (If you used the `--noauth_local_webserver` option, then `appcfg.py` will instead show you a URL to copy/paste into your browser.) Log in if necessary. The page will ask whether you wish to give appcfg access. Click OK. (If you used the `--noauth_local_webserver` option, then you will be given a token that you will need to supply to the prompt from `appcfg.py`.)

appcfg.py continues:

```
Authentication successful.

Scanning files on local disk.

...more of the usual output...
```

From now on, when you run `appcfg.py --oauth2`, it uses the saved authentication token.

If you have some automated application managment, you might want `appcfg.py` to run non-interactively. To use OAuth2 without interacting with the script, you can pass the `--oauth2_refresh_token` to `appcfg.py`:

```
appcfg.py --oauth2_refresh_token=token update myapp/
```

To get a refresh token, you can look for a `.appcfg_oauth2_tokens` file in your home directory on a machine where you have set up OAuth2. This file is in JSON format; you want the `refresh_token` value.

Command-line arguments

The `appcfg.py` command takes a set of options, an action, and arguments for the action.

The following actions are available:

`appcfg.py [options] update <app-directory>|<files...>`

> If you specify an app-directory, it must contain of the files required by the app. The update command creates or updates the app version named in the app.yaml file at the top level of the directory. It follows symlinks and recursively uploads all files to the server. Temporary or source control files (e.g. foo~, .svn/*) are skipped.

`appcfg.py help <action>`

Prints a help message about the given action, then quits.

`appcfg.py [options] cron_info <app-directory>`

Displays information about the scheduled task (cron) configuration, including the expected times of the next few executions. By default, displays the times of the next 5 runs. You can modify the number of future run times displayed with the `--num_runs=...` option.

`appcfg.py help <action>`

Prints a help message about the given action, then quits.

`appcfg.py download_app -A <app_id> -V <version><output-dir>`

Retrieves the most current version of your application's code using the following options:

`-A`
: The application ID (required).

`-V`

: The current application version. May be one of the following:

  - hi`-V major.minor` – Specifies the exact version specified.
  - `-V major` – Specifies the latest major version.
  - Omitted entirely – Returns the current default version, if one exists

`<output-dir>`

: The directory where you wish to save the files (required).

`appcfg.py [options] request_logs <app-directory> <output-file>`

Retrieves log data for the application running on App Engine. `output-file` is the name of the file to create, replace or append (if the `--append` flag is set). If `output-file` is a hyphen (`-`), the log data is printed to the console. The following options apply to `request_logs`:

`--num_days=...`
: The number of days of log data to retrieve, ending on the current date at midnight UTC. A value of 0 retrieves all available logs. If `--append` is given, then the default is 0, otherwise the default is 1.

`--end_date=...`
: The latest date of log data to return, in the form `YYYY-MM-DD`. The default is today. The `--num_days` option counts backwards from the end date.

`--severity=...`
: The minimum log level for the log messages to retrieve. The value is a number corresponding to the log level: 4 for CRITICAL, 3 for ERROR, 2 for WARNING, 1 for INFO, 0 for DEBUG. All messages at the given log level and above will be retrieved. Default is 1 (INFO).

`--append`

Append the fetched data to the output file, starting with the first log line not already present in the file. Running this command once a day with `--append` results in a file containing all log data.

The default is to overwrite the output file. Does not apply if `output-file` is `-` (printing to the console).

`--vhost=...`

If present, limits the log messages downloaded to just those emitted by requests for a given domain name. For instance, `--vhost=example.com` will download just the log messages for the live app at the Google Apps domain `example.com`, excluding any log messages emitted by a new version being tested at `http://2.latest.your_app_id.appspot.com`. Similarly, `--vhost=2.latest.your_app_id.appspot.com` downloads just the logs for the test version, excluding the live version.

The value is a regular expression that matches the `Host` header of the incoming requests. Note that the pattern is case sensitive, even though domain names usually are not.

`--include_vhost`

Include the domain name for each request (the `Host` request header) in the request log data, as an additional field.

`appcfg.py [options] rollback <app-directory>`

Undoes a partially completed update for the given application. You can use this if an update was interrupted, and the command is reporting that the application cannot be updated due to a lock.

`appcfg.py [options] set_default_version <app-directory>`

Sets the default (serving) version of the app. By default, the serving version is set to the version specified in app.yaml, unless you specify another with the `--version` option.

`appcfg.py [options] update <app-directory>`

Uploads files for an application given the application's root directory. The application

ID and version are taken from the `app.yaml` file in the app directory. The following option applies to `update`:

`--max_size`
: Maximum size of a file to upload (in bytes).

`appcfg.py [options] update_cron <app-directory>`

Updates the schedule task (cron) configuration for the app, based on the `cron.yaml` file.

`appcfg.py [options] update_dos <app-directory>`

Updates the DoS Protection configuration for the app, based on the `dos.yaml` file.

`appcfg.py [options] update_queues <app-directory>`

Updates the task queue configuration for the app, based on the `queue.yaml` file.

`appcfg.py [options] start <app-directory>`

Start the specified module version.

`appcfg.py [options] stop <app-directory>`

Stop the specified module version.

The `appcfg.py` command accepts the following options for all actions:

`--application=...`

The application ID to use. By default, this is derived from the `application:` line in the `app.yaml` file in the application directory. If `--application` is specified, it overrides the ID in `app.yaml` for this action.

`--email=...`

The email address of the Google account of an administrator for the application, for actions that require signing in. If omitted and no cookie is stored from a previous use of the command, the command will prompt for this value.

`--host=...`

The hostname of the local machine for use with remote procedure calls.

`--max_size=...`

A maximum size of files to upload, as a number of bytes. Files larger than this size will not be uploaded. The default is `10485760`. The server currently enforces a maximum file size of 10,485,760 bytes, so increasing this value will not have any effect.

`--no_cookies`

Do not store the administrator sign-in credentials. Prompt for a password every time (or go through the OAuth2 flow if the `--oauth2` options is used).

`--noisy`

Print many messages about what the command is doing. This is mostly useful when working with the App Engine team to troubleshoot an upload issue.

`--noauth_local_webserver`

Don't use the web browser to display web pages for authentication. This is useful for OAuth2 authentication if you're remotely logged on to a machine. This flag only applies for OAuth2.

`--oauth2`

Use OAuth2 authentication instead of password-based authentication.

`--oauth2_refresh_token=token`

Use an OAuth2 refresh token to authenticate instead of password-based authentication or a stored token.

`--passin`

If given, the tool accepts the Google account password in stdin instead of prompting for it interactively. This allows you to invoke the tool from a script without putting your password on the command line.

`--quiet`

Do not print messages when successful.

`--server=...`

The App Engine server hostname. The default is `appengine.google.com`.

`--verbose`

Print messages about what the command is doing.

`--version=...`

The version ID to use. By default, this is derived from the `version:` line in the `app.yaml` file in the application directory. If `--version` is specified, it overrides the version ID in `app.yaml` for this action.