GOOGLE APP ENGINE

CHAPTER 8

PHP: SERVICE APIs

PHP Service APIs

This section documents the services available in the PHP App Engine runtime. Some services are available in all runtime languages, others only in a subset of languages. While the functionality of a service is usually constant across all runtimes that offer it, there can be exceptions.

App Identity PHP API Overview

Code sometimes needs to determine the identifier of the application in which it is executing. This may be to generate a URL or email address, or possibly to make some run-time decision. App Engine includes an Application Identity service for this purpose.

Identifying itself

Application ID

The application ID can be found using the `getApplicationId()` method.

Versioned hostnames

A related operation is the need to get the hostname part of a URL to the application. You can use the`getDefaultVersionHostname()` method for this purpose. This is useful in certain scenarios when the application is not available at `http://your_app_id.appspot.com`.

Asserting identity to other App Engine apps

If you want to determine the identity of the App Engine app that is making a request to your App Engine app, you can use the request header `X-Appengine-Inbound-Appid`. This header is added to the request by the URLFetch service and is not user modifiable, so it safely indicates the requesting application's ID, if present.

In order for this header to be added to the request, the app making the request must tell the UrlFetch service to not follow redirects when it invokes URLFetch. App Engine will then automatically add the header to the HTTP response.

In your application handler, you can check the incoming ID by reading the `X-Appengine-Inbound-Appid`header and comparing it to a list of IDs allowed to make requests.

Asserting identity to Google APIs

Many Google APIs support OAuth assertions to identify the source of the request. The App Identity API provides a service that creates tokens that can be used to assert that the source of a request is the application itself. The`getAccessToken()` method returns an access token for a scope, or list of scopes. This token can then be set in the HTTP headers of a call to identify the calling application.

The following illustrates how to use the App Identity API to retrieve Google Calendar contacts using OAuth.

```
// Retrieves Google Calendar contacts using OAuth
use google\appengine\api\app_identity\AppIdentityService;
function setAuthHeader() {
  $access_token =
AppIdentityService::getAccessToken('https://www.google.com/m8/feeds');
  return [sprintf('Authorization: OAuth %s', $access_token['access_token'])];
}
$get_contacts_url = 'https://www.google.com/m8/feeds/contacts/default/full';
$headers = setAuthHeader();
$opts = [
  'http' => [
    'header' => implode("\r\n", $headers),
  ],
];
$context = stream_context_create($opts);
$response = file_get_contents($get_contacts_url, false, $context);
$xml = simplexml_load_string($response);
$email = $xml->author->email;
$service_account = AppIdentityService::getServiceAccountName();
if (strcmp($email, $service_account) != 0) {
  die(sprintf('%s does not match the service account name %s.',
              $email,
              $service_account));
}
```

Note that the application's identity is represented by the service account name, which is typically*applicationid@appspot.gserviceaccount.com*. You can get the exact value by using the`getServiceAccountName()` method. For services which offer ACLs, you can grant the application access by granting this account access.

Asserting identity to other systems

The token generated by `getAccessToken()` only works against Google systems. However you can use the underlying signing technology to assert the identity of your application to other systems. The `signForApp()`method will sign bytes using a private key unique to your application, and the `getPublicCertificates()`method will return certificates which can be used to validate the signature.

Note: The certificates may be rotated from time to time, and the method may return multiple certificates, all of which are currently valid.

Logs PHP API Overview

Overview

The Logs API provides access to the application and request logs for your application. You can also access the logs for your application in the Logs Viewer provided in the Google Developers Console by clicking Monitoring > Logs in the left navigation panel

Note: Currently, the Logs API only generates output for deployed apps.

The Logs PHP API is invoked whenever the built-in PHP function`syslog()` is called. For example:

```
if (authorizedUser()) {

  echo '<p>Welcome authorized user</p>';

  syslog(LOG_INFO, 'Authorized access');

} else {

  echo 'Go away unauthorized user<p />';

  syslog(LOG_WARNING, "Unauthorized access");

}
```

You do not need to call `openlog()` or `closelog()` before writing to `syslog`. Calls to these functions will be ignored.

Log categories: request logs and app logs

There are two categories of log data: request logs and application logs. A request log is written for each request handled by your app, and contains information such as the app ID, HTTP version, and so forth.

Getting log data

The general process of getting logs is as follows:

1. Use `fetch()` to return an iterator for the request logs.
2. In each iteration, process each `RequestLog` as desired.

3.Optionally, use `getAppLogs()` to get the list of related AppLogs.

4.If you retrieved the app logs list, for each `AppLogLine`, process the `AppLog` property data as desired.

Sample code

The following sample reads request logs between given start and end times and verifies that the given messages and levels are contained sequentially in the app logs. The response will be 'PASS' if all expected messages and levels are matched against app logs.

```
use google\appengine\api\log\LogService;
// LogService API usage sample to display application logs for last 24 hours.
$options = [
  // Fetch last 24 hours of log data
  'start_time' => (time() - (24 * 60 * 60)) * 1e6,
  // End time is Now
  'end_time' => time() * 1e6,
  // Include all Application Logs (i.e. your debugging output)
  'include_app_logs' => true,
  // Filter out log records based on severity
  'minimum_log_level' => LogService::LEVEL_INFO,
];
$logs = LogService::fetch($options);
foreach ($logs as $log) {
  echo '<ul>REQUEST LOG';
  echo '<li>IP: ', $log->getIp(), '</li>',
         '<li>Status: ', $log->getStatus(), '</li>',
         '<li>Method: ', $log->getMethod(), '</li>',
         '<li>Resource: ', $log->getResource(), '</li>';
  $end_date_time = $log->getEndDateTime();
  echo '<li>Date: ',$end_date_time->format('c'), '</li>';
  $app_logs = $log->getAppLogs();
  foreach ($app_logs as $app_log) {
    echo '<ul>APP LOG';
    echo '<li>Message: ', $app_log->getMessage(), '</li>';
    $app_log_date_time = $app_log->getDateTime();
    echo '<li>Date: ', $app_log_date_time->format('c'), '</li></ul>';
  }
  echo '</ul>';
}
```

How to read a log

Note: the logs described here are shown as visible in the Log Viewer.

To view logs using the Log Viewer:

1.Visit the developer console in your browser.

2.Open the project whose logs you wish to see and select Compute > App Engine > Logs.

3.Use the desired filter to retrieve the logs you want to see. You can filter by various combinations of time, log level, module, and log filter label or regular expression.

Notice that labels are regular expressions for filtering the logs by logging fields. Valid labels include the following:

- day
- month
- year
- hour
- minute
- second
- tzone
- remotehost
- identd_user
- user
- status
- bytes
- referrer
- useragent
- method
- path
- querystring
- protocol
- request_id

For example, `path:/foo.* useragent:.*Chrome.*` gets logs for all requests to a path starting with `/foo` that were issued from a Chrome browser.

A typical App Engine log contains data in the Apache combined log format, along with some special App Engine fields, as shown in the following sample log:

```
192.0.2.0 test [27/Jun/2014:09:11:47 -0700] "GET / HTTP/1.1" 200 414 -

"http://www.example.com/index.html"

Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/35.0.1916.153 Safari/537.36"

"1-dot-calm-sylph-602.appspot.com" ms=195 cpu_ms=42 cpm_usd=0.000046

loading_request=1 instance=00c61b117cfeb66f973d7df1b7f4ae1f064d
app_engine_release=1.9.15
```

The following table lists the fields in order of occurrence along with a description:

Field Order	Field Name	Always Present?	Description
1	Client address	Yes	Client IP address. Example: `192.0.2.0`
2	RFC1413 identity	No	RFC1413 identity of the client. This is nearly always the character `-`
3	User	No	Present only if the app uses the Users API and the user is logged in. This value is the "nickname" portion of the Google Account, for example, if the Google Account is `test@example.com`, the nickname that is logged in this field is `test`.
4	Timestamp	Yes	Request timestamp. Example: `[27/Jun/2014:09:11:47 -0700]`
5	Request querystring	Yes	First line of the request, containing method, path, and HTTP version. Example: `GET / HTTP/1.1`
6	HTTP Status Code	Yes	Returned HTTP status code. Example: `200`
7	Response size	Yes	Response size in bytes. Example: `414`
8	Referrer path	No	If there is no referrer, the log contains no path, but only `-`. Example referrer path: `"http://www.example.com/index.html"`.
9	User-agent	Yes	Identifies the browser and operating system to the web server. Example:`Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36`
10	Hostname	Yes	The hostname used by the client to connect to the App Engine application. Example : (`1-dot-calm-sylph-602.appspot.com`)
11	Wallclock time	Yes	Total clock time in milliseconds spent by App Engine on the request. This time duration does not include time spent between the client and the server running the instance of your application. Example: `ms=195`.
12	CPU milliseconds	Yes	CPU milliseconds required to fulfill the request. This is the number of milliseconds spent by the CPU actually executing your application code, expressed in terms of a baseline 1.2 GHz Intel x86 CPU. If the CPU actually used is faster than the baseline, the CPU milliseconds can be larger than the actual

Field Order	Field Name	Always Present?	Description
			clock time defined above. Example: `cpu_ms=42`
13	Exit code	No	Only present if the instance shut down after getting the request. In the format `exit_code=XXX` where `XXX` is a 3 digit number corresponding to the reason the instance shut down. The exit codes are not documented since they are primarily intended to help Google spot and fix issues.
14	Estimated cost	Yes	Estimated cost of 1000 requests just like this one, in USD. Example:`cpm_usd=0.000046`
15	Queue name	No	The name of the task queue used. Only present if request used a task queue. Example: `queue_name=default`
16	Task name	No	The name of the task executed in the task queue for this request. Only present if the request resulted in the queuing of a task. Example:`task_name=7287390692361099748`
17	Pending queue	No	Only present if a request spent some time in a pending queue. If there are many of these in your logs and/or the values are high, it might be an indication that you need more instances to serve your traffic. Example:`pending_ms=195`
18	Loading request	No	Only present if the request is a loading request. This means an instance had to be started up. Ideally, your instances should be up and healthy for as long as possible, serving large numbers of requests before being recycled and needing to be started again. Which means you shouldn't see too many of these in your logs. Example: `loading_request=1`.
19	Instance	Yes	Unique identifier for the instance that handles the request. Example:`instance=00c61b117cfeb66f973d7df1b7f4ae1f064d`
20	Version	Yes	The current App Engine release version used in production App Engine: 1.9.15

Quotas and limits

Your application is affected by the following logs-related quotas:

- Logs data retrieved via the Logs API.
- Log storage, also called *logs retention*.

Quota for data retrieved

The first 100 megabytes of logs data retrieved per day via the Logs API calls are free. After this amount is exceeded, no further Logs API calls will succeed unless billing is enabled for your app. If billing is enabled for your app, data in excess of 100 megabytes results in charges of $0.12/GB.

Logs storage

You can control how much log data your application stores by means of its log retention settings in the Admin Console. By default, logs are stored for an application free of charge with the following per-application limits: a maximum of 1 gigabyte for a maximum of up to 90 days. If either limit is exceeded, more recent logs will be shown and older logs will be deleted to stay within the size limit. Logs older than the maximum retention time are also deleted.

If your app has billing enabled, you can pay for higher log size limits by specifying the desired maximum log sizein gigabytes in the Admin Console. You can also set the retention time by specifying the desired number of days to keep logs, up to a maximum of 365 days. The cost of this extra log storage is $0.026 per gigabyte utilized per month.

Limit	Amount	Cost past free threshold
Maximum days storage per log	90 days free, 365 days if paid	$0.026 per gigabyte utilized per month
Maximum total logs storage	1 gigabyte free, unlimited if paid	$0.026 per gigabyte utilized per month

Mail PHP API Overview

Sending mail

The Mail service can send email messages to one or more recipients. The message contains a subject, a plaintext body, and an optional HTML body. It can also contain file attachments, as well as a limited set of headers.

For security purposes, the sender address of a message must be the email address of an administrator for the application or any valid email receiving address for the app (see Receiving Mail). The sender can also be the Google Account email address of the current user who is signed in, if the user's account is a Gmail account or is on a domain managed by Google Apps.

If you want to send email on behalf of the application but do not want to use a single administrator's personal Google Account as the sender, you can create a new Google Account for the application using any valid email address, then add the new account as an administrator for the application. To add an account as an administrator, see the "Permissions" section of the Admin Console. Accounts must be given "Owner" or "Developer" level access when added.

You can also send mail using a domain account by adding the domain account under "Permissions" in the Admin Console. Domain accounts are accounts outside of the Google domain with email addresses that do not end in @gmail.com or @APP-ID.appspotmail.com. You should set the SPF records for your domain to indicate that Google is a trusted source for your email. For instructions on how to do this, see SPF records in the Google Apps help articles.

Note: Your domain (e.g. example.com) needs to be explictly registered with Google Apps and verified before you can create and use domain accounts (e.g. user@example.com). Domain accounts do not need to be explicitly verified, since you will have verified the domain during the registration process. For more information about registering a domain, see Register a new domain.

You can use any email address for a recipient. A recipient can be in the message's "to" field or the "cc" field, or the recipient can be hidden from the message header (a "blind carbon copy" or "bcc").

When an application calls the Mail service to send a message, the message is queued and the call returns immediately. The Mail service uses standard procedures for contacting each recipient's mail server, delivering the message, and retrying if the mail server cannot be contacted.

If the Mail service cannot deliver a message, or if an recipient's mail server returns a

bounce message (such as if there is no account for that address on that system), the error message is sent by email to the address of the sender for the message. The application itself does not receive any notification about whether delivery succeeded or failed.

PHP's built-in `mail()` function has been disabled in the PHP runtime for Google App Engine. Instead, to send a mail, you must make direct calls to the App Engine mail API, as shown in this example snippet:

```
use \google\appengine\api\mail\Message;

$image_content_id = '<image-content-id>';

try

{

  $message = new Message();

  $message->setSender("from@google.com");

  $message->addTo("to@google.com");

  $message->setSubject("Example email");

  $message->setTextBody("Hello, world!");

  $message->addAttachment('image.jpg', 'image data', $image_content_id);

  $message->send();

} catch (InvalidArgumentException $e) {

  // ...

}
```

Receiving mail

Your app can receive email at addresses of the following form:

```
string@appid.appspotmail.com
```

Note that even if your app is deployed on a custom domain, your app can't receive email sent to addresses on that domain.

Email messages are sent to your app as HTTP requests. These requests are generated by App Engine and posted to your app. In your app's configuration, you specify handlers that will be called to handle these HTTP requests. The handlers run in the default module (or

application version). They receive the MIME data for email messages, which you then parse into its individual fields.

Email messages are sent to your app as HTTP POST requests using the following URL:

```
/_ah/mail/address
```

where `address` is a full email address, including domain name.

To receive email, you first edit your app's configuration file to include a section that enables incoming mail:

```
inbound_services:

- mail
```

Incoming email in App Engine works by posting HTTP requests containing MIME data to your app. In your configuration file, you must create mappings from URL paths that represent email addresses to handlers in your app's code. The pattern `/_ah/mail/.+` matches all incoming email addresses:

```
- url: /_ah/mail/.+

  script: handle_incoming_email.php

  login: admin
```

In the app itself, you must implement code in the handlers you specified. The email's MIME data is supplied to your app as the contents of an HTTP POST request, and you process this data in your handlers.

Receiving bounce notification

In order to receive email bounce notifications, you need to configure your app to enable email notification and you need to create a handler to handle those notifications.

There are two parts to the configuration. First, you need to enable notification. Second, you need to set the mapping between `/_ah/bounce` and your bounce handler, so App Engine knows where to POST the notification data. To configure your app to receive bounced email notifications:

1.Add the following to your `app.yaml` file to enable notification:

```
inbound_services:

- mail_bounce
```

2.Also in `app.yaml`, declare a mapping between `/_ah/bounce` and the bounce notification handler in your code, for example:

```
- url: /_ah/bounce

  script: handle_bounced_email.php

  login: admin
```

In your app, you'll need to supply bounce handler code to receive and process the notifications. Your application will recieve a HTTP POST with the body of the message being the bounce notification for the mail message.

Sending mail with attachments

An outgoing email message can have zero or more file attachments.

An attachment has a filename and file data. The file data can come from any source, such as an application data file or the datastore. The MIME type of the attachment is determined from the filename.

The following is a list of MIME types and their corresponding filename extensions allowed for file attachments to an email message. You are not limited to these extensions. If you use an unknown extension, App Engine will assign it the mime type `application/octet-stream`.

MIME Type	Filename Extension(s)
application/msword	doc
application/msword	docx
application/pdf	pdf
application/rss+xml	rss

MIME Type	Filename Extension(s)
application/vnd.google-earth.kml+xml	kml
application/vnd.google-earth.kmz	kmz
application/vnd.ms-excel	xls
application/vnd.ms-excel	xlsx
application/vnd.ms-powerpoint	pptx
application/vnd.ms-powerpoint	pps ppt
application/vnd.oasis.opendocument.presentation	odp
application/vnd.oasis.opendocument.spreadsheet	ods
application/vnd.oasis.opendocument.text	odt
application/vnd.sun.xml.calc	sxc
application/vnd.sun.xml.writer	sxw
application/x-gzip	gzip
application/zip	zip
audio/basic	au snd
audio/flac	flac
audio/mid	mid rmi
audio/mp4	m4a
audio/mpeg	mp3
audio/ogg	oga ogg
audio/x-aiff	aif aifc aiff
audio/x-wav	wav
image/gif	gif
image/jpeg	jpeg jpg jpe
image/png	png
image/tiff	tiff tif
image/vnd.wap.wbmp	wbmp
image/x-ms-bmp	bmp
text/calendar	ics

MIME Type	Filename Extension(s)
text/comma-separated-values	csv
text/css	css
text/html	htm html
text/plain	text txt asc diff pot
text/x-vcard	vcf
video/mp4	mp4
video/mpeg	mpeg mpg mpe
video/ogg	ogv
video/quicktime	qt mov
video/x-msvideo	avi

As a security measure to protect against viruses, you cannot send email attachments or zip files containing any of the following extensions:

- .ade
- .adp
- .bat
- .chm
- .cmd
- .com
- .cpl
- .exe
- .hta
- .ins
- .isp
- .jse
- .lib
- .mde
- .msc
- .msp
- .mst
- .pif
- .scr
- .sct
- .shb
- .sys
- .vb
- .vbe
- .vbs
- .vxd
- .wsc
- .wsf
- .wsh

Sending mail with headers

An outgoing email can have zero or more extra headers. A header has a name and a value.

For security purposes, the name of a header must be of one of the allowed header names:

- In-Reply-To
- List-Id
- List-Unsubscribe
- On-Behalf-Of
- References
- Resent-Date
- Resent-From
- Resent-To

Mail and the development server

When an application running in the development server calls the Mail service to send an email message, the message is printed to the log. The PHP development server does not send the email message.

Authenticating mail: DKIM

If your application sends messages from an email address that is part of a Google Apps domain, App Engine can utilize a Google Apps feature to cryptographically sign the emails it sends. This signature says that this mail that purports to be from `emma@example.com` really came from `example.com`. The recipient can check this signature; if the signature is there and correct, the recipient knows that the sender's domain wasn't spoofed. App Engine uses the DomainKeys Identified Mail (DKIM) standard to authenticate the sender's domain.

To enable DKIM authentication for messages sent from Google Apps email addresses, follow these instructionsin the Google Apps Help Center. Note that it may take up to 48 hours before DKIM authentication is active for your Google Apps domain.

App Engine will sign the application's outgoing mails if the sender address is part of a Google Apps domain with DKIM enabled. Additionally, the sender address must be formatted such that the domain part of the email address only consists of lowercase letters.

Bulk senders guidelines

You must follow the guidelines in this section if your application is sending out bulk email, i.e. similar messages to numerous recipients. These guidelines will help to improve your inbox delivery rate to Gmail users, by ensuring that all recipients in your distribution list actually want to receive the email. If recipients manually mark your email as spam then that acts as a strong signal to Gmail to mark future emails from you as spam.

Authentication and identification

•Use the same sender for every bulk email. When calling the Mail API function to send email, the `From`header will be set to match the sender you specify.

•Your sender address should be an account in a Google Apps for Business domain. Google accounts that send too many emails that are marked as spam by Google, can be temporarily disabled if their domain is still in the free trial period or has less than six users. In these cases, the Mail API will throw an exception with an `Unauthorized sender` error message.

•Sign your email with DKIM, which requires a Google Apps domain if you are sending using App Engine.

•Publish an SPF record to prevent spammers from spoofing your envelope sender address. SPF verifies that email is sent from an IP address that is published in the DNS records of the envelope sender. App Engine's envelope sender is in the `apphosting.bounces.google.com` domain, so your SPF record may not be used to determine if email from App Engine should be delivered.

Sending limits

•Your Mail quota is shown in the Quota Details page in the Admin Console. The quota is reset daily. You will get an over quota exception if you exceed the daily quota. See the Quotas and Limits section for more details. To request a quota increase, go to the Quotas documentation page.

•You should throttle sending of emails to avoid sending too many emails in a short burst, which could cause some emails to be silently dropped due to a safety limit on Google's side. You can calculate the maximum daily rate of sending emails per second by dividing your daily quota by 86,400, the number of seconds in a day. We recommend that you do not send bulk email with short bursts at higher than 50 times this long term rate.

Subscription

•Each user in your distribution list should opt-in to receive messages from you in one of the following ways:

- •By sending you an email asking to subscribe
- •By manually checking a box on a web form, or within a piece of software

•Using an email address list purchased from a third-party is not considered opt-in. You also should not set a checkbox on a web form or within a piece of software to subscribe all users by default. Users should not be required to explicitly opt-out of mailings.

•You should verify that the person that signed up by checking the box on the web form or in software is actually receiving emails at the address that was specified in the form, by sending an email that requires them to confirm receipt.

Unsubscribing

•A user must be able to unsubscribe in one of the following ways:

- •Through a prominent link in the email with no further user interaction other than confirmation
- •Via an email unsubscribe response

•App Engine can only receive email sent to the `appid.appspotmail` domain. Therefore, you will need to set your sender to an address in this domain if you want to automatically handle email unsubscribe responses within App Engine.

•Use the `List-Unsubscribe` header, which is supported by the App Engine Mail API.

•Automatically unsubscribe users whose addresses bounce multiple pieces of email. You can configure your app to receive bounce notifications.

•Periodically send email confirmations to users, offering the opportunity to unsubscribe from each list they are signed up for.

•You should explicitly indicate the email address subscribed within your email because users may forward email from other accounts.

Format

•Format to RFC 2822 SMTP standards and, if using HTML, w3.org standards.

•Attempts to hide the true sender of the message or the true landing page for any web links in the message may result in non-delivery. For example, we recommend that you do not use URL shortener services in bulk email, since these can mask the real URLs contained in the body of your email.

•The subject of each message should be relevant to the body's content and not be misleading.

Delivery

•The following factors will help messages arrive in Gmail users' inboxes:

  •The `From` address is listed in the user's Contacts list.
  •A user clicks "Not Spam" to alert Gmail that messages sent from that address are solicited.

•If you send both promotional email and transactional email relating to your organization, we recommend separating email by purpose as much as possible. You can do this by:

  •Using separate email addresses for each function.
  •Sending email from different domains for each function.

Third-party senders

•If others use your service to send email, you are responsible for monitoring your users and/or clients' behavior. You must terminate, in a timely fashion, all users and/or clients who use your service to send spam email. The Google Cloud Platform Acceptable Use Policy specifically prohibits spam. Your application can be suspended if you violate this policy, as described in the Google Cloud Platform Terms of Service.

•You must have an email address available for users and/or clients to report abuse, which should normally be `abuse@yourdomain.com`. You should also monitor `postmaster@yourdomain.com`.

•Monitor email sent to app admins. Google may need to urgently contact app admins, for example to notify you of a violation of the Acceptable Use Policy. We can help you to resolve the problems more quickly if you respond promptly to our emails.

•You must maintain up-to-date contact information in your WHOIS record maintained by your domain registrar, and on abuse.net.

Affiliate marketing programs

•Affiliate marketing programs reward third-parties for bringing visitors to your site. These programs are attractive to spammers and can potentially do more harm than good. Please note the following:

•If your brand becomes associated with affiliate marketing spam, it can affect the email sent by you and your other affiliates.
•It is your responsibility to monitor your affiliates and remove them if they send spam.

Alternatives to the App Engine Mail API

•You can use a third-party email delivery service provider to send email from App Engine. These services may provide additional features that are not available in the Mail API and may be a better solution for some bulk email senders.

Quotas and limits

Each Mail service request counts toward the Mail API Calls quota.

Each recipient email address for an email message counts toward the Recipients Emailed (billable) quota. Each recipient that is an administrator for the application also counts toward the Admins Emailed quota.

Data sent in the body of an email message counts toward the following quotas:

•Outgoing Bandwidth (billable)

•Message Body Data Sent

Each attachment included with an email message counts toward the Attachments Sent quota.

Data sent as an attachment to an email message counts toward the following quotas:

•Outgoing Bandwidth (billable)

•Attachment Data Sent

For more information on quotas, see Quotas, and the "Quota Details" section of the Admin Console.

Note: Paid applications must pay their first weekly bill before they can send mail messages beyond

the free quota. Once the first charge is cleared, each message sent above the free quota is charged at the normal paid rate.

In addition to quotas, the following limits apply to the use of the Mail service:

Limit	Amount
maximum size of outgoing mail messages, including attachments	10 megabytes
maximum size of incoming mail messages, including attachments	10 megabytes
maximum size of message when an administrator is a recipient	16 kilobytes

Memcache PHP API Overview

High performance scalable web applications often use a distributed in-memory data cache in front of or in place of robust persistent storage for some tasks. App Engine includes a memory cache service for this purpose.

Note: The cache is global and is shared across the application's frontend, backend, and all of its modules/versions.

PHP memcache implementation

App Engine includes implementations of the standard Memcacheand Memcached APIs, which invoke the App Engine memcache service "under the hood". Some functions are callable ("stubbed") but do nothing, as they aren't needed in the context of an App Engine app. As such, calls to the following functions will be ignored:

Stubbed functions in the Memcache API

- `memcache_add_server()`
- `memcache_close()`
- `memcache_connect()`
- `memcache_pconnect()`
- `memcache_set_compress_threshold()`
- `addServer()`
- `close()`
- `connect()`
- `pconnect()`
- `setCompressThreshold()`

Stubbed functions in the Memcached API

- `addServer()`
- `addServers()`
- `getAllKeys()`
- `getServerByKey()`
- `getServerList()`
- `getStats()`
- `getVersion()`
- `isPersistent()`
- `isPristine()`
- `quit()`
- `resetServerList()`
- `setSaslAuthData()`

An example usage of the Memcache PHP API in App Engine:

```
function getData($key) {
  $memcache = new Memcache;
  $data = $memcache->get($key);
  if ($data === false) {
    $data = doSlowQuery($key);
    $memcache->set($key, $data);
  }
  return $data;
}
```

An example usage of the Memcached PHP API in App Engine:

```
$memcache = new Memcached;
$memcache->setMulti(['image' => $image, 'data' => $data], 300);
$memcache->increment('hits');
```

When to use a memory cache

One use of a memory cache is to speed up common datastore queries. If many requests make the same query with the same parameters, and changes to the results do not need to appear on the web site right away, the app can cache the results in the memcache. Subsequent requests can check the memcache, and only perform the datastore query if the results are absent or expired. Session data, user preferences, and any other queries performed on most pages of a site are good candidates for caching.

Memcache may be useful for other temporary values. However, when considering whether to store a value solely in the memcache and not backed by other persistent storage, be sure that your application behaves acceptably when the value is suddenly not available. Values can expire from the memcache at any time, and may be expired prior to the expiration deadline set for the value. For example, if the sudden absence of a user's session data would cause the session to malfunction, that data should probably be stored in the datastore in addition to the memcache.

How cached data expires

By default, values stored in memcache are retained as long as possible. Values may be evicted from the cache when a new value is added to the cache if the cache is low on memory. When values are evicted due to memory pressure, the least recently used values are evicted first.

The app can provide an expiration time when a value is stored, as either a number of seconds relative to when the value is added, or as an absolute Unix epoch time in the future (a number of seconds from midnight January 1, 1970). The value will be evicted no later than this time, though it may be evicted for other reasons.

Under rare circumstances, values may also disappear from the cache prior to expiration for reasons other than memory pressure. While memcache is resilient to server failures, memcache values are not saved to disk, so a service failure may cause values to become unavailable.

In general, an application should not expect a cached value to always be available.

You can erase an application's entire cache via the API or via the Admin Console (under Memcache Viewer).

Statistics

Memcache maintains statistics about the amount of data cached for an application, the cache hit rate, and the age of cache items. You can view these statistics using the API or in the Administration Console, under Memcache Viewer.

Limits

The following limits apply to the use of the memcache service:

- The maximum size of a cached data value is 1 MB minus the size of the key minus an implementation-dependent overhead which is approximately 96 bytes.

- A key cannot be larger than 250 bytes. In the PHP runtime, if you try to set memcache with a larger key the call will raise an exception. (Other runtimes behave differently.)

- The "multi" batch operations can have any number of elements. The total size of the call and the total size of the data fetched must not exceed 32 megabytes.

Configuring memcache

The memcache service provides best-effort cache space by default. Apps with billing enabled may opt to use dedicated memcache which provides a fixed cache size assigned exclusively to your app. The service is configured via memcache settings on the Admin Console.

Sockets PHP API Overview

Note: Sockets are only available for paid apps, and traffic from sockets is billed as outgoing bandwidth. Sockets are also limited by daily and per minute (burst) quotas. App Engine supports outbound sockets using methods from the standard PHP library such as`fsockopen`. For supported options, calls to `socket_get_option`will return a mock value and calls to `socket_set_option` will be silently ignored. Errors will continue to be raised for unsupported options.

The currently supported options are:

- `SO_KEEPALIVE`
- `SO_LINGER`
- `SO_RCVBUF`
- `SO_DEBUG`
- `SO_OOBINLINE`
- `SO_REUSEADDR`
- `TCP_NODELAY`
- `SO_SNDBUF`

Limitations and restrictions

App Engine supports sockets without requiring you to import any special App Engine libraries or add any special App Engine code. However, there are certain limitations and behaviors you need to be aware of when using sockets:

- Sockets are available only for paid apps.
- You cannot create a listen socket; you can only create outbound sockets.
- You can only use TCP or UDP; arbitrary protocols are not allowed.
- You cannot bind to specific IP addresses or ports.
- Port 25 (SMTP) is blocked; you can still use authenticated SMTP on the submission port 587.
- Private, broadcast, multicast, and Google IP ranges (except those whitelisted below), are blocked:
  - Google Public DNS: `8.8.8.8`, `8.8.4.4`, `2001:4860:4860::8888`, `2001:4860:4860::8844` port 53
  - Gmail SMTPS: `smtp.gmail.com` port 465 and 587
  - Gmail POP3S: `pop.gmail.com` port 995

- Gmail IMAPS: `imap.gmail.com` port 993

- Socket descriptors are associated with the App Engine app that created them and are non-transferable (cannot be used by other apps).

- Sockets may be reclaimed after 2 minutes of inactivity; any socket operation keeps the socket alive for a further 2 minutes.

Using sockets with the development server

You can run and test code using sockets on the development server, without using any special command line parameters.

SMS and Voice Services via Twilio

Note: Twilio is a third-party company whose services are not covered by the Google App Engine Service Level Agreement.

Twilio is powering the future of business communications, enabling developers to embed voice, VoIP, and messaging into applications. They virtualize all infrastructure needed in a cloud-based, global environment, exposing it through the Twilio communications API platform. Applications are simple to build and scalable. Enjoy flexibility with pay-as-you go pricing, and benefit from cloud reliability.

Twilio Voice enables your application to make and receive phone calls. Twilio SMS enables your application to send and receive text messages. Twilio Client allows you to make VoIP calls from any phone, tablet, or browser and supports WebRTC.

Pricing

Google App Engine customers receive complimentary credit for 2,000 SMS messages or inbound minutes when you upgrade. Redeem this Twilio credit and get started here.

Twilio is a pay-as-you-go service. There are no set-up fees and you can close your account at any time. You can find more details at Twilio Pricing.

Platform

The Twilio platform consists of the Twilio Markup Language (TwiML), a RESTful API and VoIP SDKs for web browsers, Android and iOS. Helper libraries are available in multiple languages. You can find the full list at Twilio Helper Libraries.

TwiML

TwiML is a set of instructions you can use to tell Twilio what to do when you receive an incoming call or SMS. When someone makes a call or sends an SMS to one of your Twilio numbers, Twilio will look up the URL associated with that phone number and make a request to that URL. Twilio will read TwiML instructions at that URL to determine what to do:

- `<Say>` - text to speech
- `<Record>` - record the call
- `<Play>` - play a message for the caller

- `<Gather>` - prompt the caller to press digits on their keypad
- `<Sms>` - send an SMS

Learn about the other verbs and capabilities from the Twilio Markup Language documentation.

REST API

The Twilio REST API allows you to query metadata about your account, phone numbers, calls, text messages, and recordings. You can also do some fancy things like initiate outbound calls and send text messages.

Since the API is based on REST principles, it's very easy to write and test applications. You can use your browser to access URLs, and you can use pretty much any HTTP client in any programming language to interact with the API. Learn more about the Twilio REST API.

Setting Up

We will be using the standard Google App Engine PHP Runtime Environment to construct this example. If this is your first time writing PHP for Google App Engine, we recommend that you use the Getting Started guide for PHP. Follow the tutorial until you have completed the "Hello, World!" guide.

After you have a working "Hello, World!" application, you will need to add Twilio's PHP library to your application.

Download and decompress this package into the `helloworld` directory you created while following the Getting Started guide:

`https://github.com/twilio/twilio-php/archive/master.zip`

Include the package as shown within your application:

```
require '/path-to-your-application/twilio-php/Services/Twilio.php'
```

You have now installed the Twilio library into your Google App Engine project.

Receiving an Incoming Call

Let's walk through creating your first application, Hello Monkey.

After completing the Getting Started guide for PHP and installing the Twilio library, modify `main.php` to look likethis example PHP file.

After updating `main.php`, follow these directions to deploy your project to App Engine: Uploading your Application.

After you've deployed your project to App Engine you will be able to send a POST request to `http://<your app>.appspot.com/twiml`, which will return the following text:

```
<?xml version="1.0" encoding="UTF-8"?>

<Response>

  <Say>Hello Monkey!</Say>

</Response>
```

Next, copy and paste the `http://<your app>.appspot.com/twiml` URL into the "Voice" URL box on the Numbers page of your Twilio Account.

Now call your Twilio number! You should hear a voice say "Hello Monkey!" in response. When you call, Twilio will fetch your URL, and execute the XML instructions listed above. Then, Twilio will hang up, because there are no more instructions.

Sending an SMS

Our twilio-php helper library makes this extremely to send an outgoing SMS using Twilio. To do this, modify your `main.php` from above to look like this example PHP file.

Note that you'll need to fill in `$AccountSid` and `$AuthToken` which are found here:https://www.twilio.com/user/account.

You will also need to change the `from` and `to` parameters to use real phone numbers. The "from" number must be a valid Twilio phone number in your account. For this example, use the phone number you called in the example above. The "to" number can be any outgoing number, your cell phone for example.

After deploying the updated code, send the SMS by putting this URL into a web browser: `http://<your app>.appspot.com/`.

Learning More about Twilio

Now that you've learned some of the basics, learn more about additional features and some best practices for building secure and scalable applications:

- Twilio Security Guidelines
- Twilio HowTo’s and Example Code
- Twilio Quickstart Tutorials
- Twilio on GitHub
- Talk to Twilio Support

URL Fetch PHP API Overview

App Engine applications can communicate with other applications or access other resources on the web by fetching URLs. An app can use the URL Fetch service to issue HTTP and HTTPS requests and receive responses. The URL Fetch service uses Google's network infrastructure for efficiency and scaling purposes.

Fetching URLs in PHP

Note: The URL Fetch service will always issue a GET request when it receives and responds to a 302 response. See more information in the release notes.

In the PHP runtime for Google App Engine, the URL Fetch service is invoked whenever you use the standard PHPwrappers for HTTP(s) URLs that are used by built-in filesystem functions such as fopen().

```
$data = ['data' => 'this', 'data2' => 'that'];

$data = http_build_query($data);

$context = [

  'http' => [

    'method' => 'GET',

    'header' => "custom-header: custom-value\r\n" .

                "custom-header-two: custom-value-2\r\n",

    'content' => $data

  ]

];

$context = stream_context_create($context);

$result = file_get_contents('http://app.com/path?query=update', false,
$context);
```

There are a few considerations to keep in mind when using these wrappers in your Google App Engine application:

- When connecting to web sites using HTTPS, the default behavior is to validate the certificate of the host and reject requests where the vertificate does not match. This is the opposite to the default behavior of PHP, and to change this behavior you can set the value of `verify_peer` to false in the ssl section of the context parameters.

- All other values of ssl contexts options are ignored.

Making requests

An app can fetch a URL using HTTP (normal) or HTTPS (secure). The URL specifies the scheme to use:`http://...` or `https://...`

The URL to be fetched can use any port number in the following ranges: 80-90, 440-450, 1024-65535. If the port is not mentioned in the URL, the port is implied by the scheme: `http://...` is port 80, `https://...` is port443.

The fetch can use any of the following HTTP methods: `GET` (common for requesting web pages and data), `POST`(common for submitting web forms), `PUT`, `HEAD`, and `DELETE`. The fetch can include HTTP request headers and a payload (an HTTP request body).

The URL Fetch service uses an HTTP/1.1 compliant proxy to fetch the result.

To prevent an app from causing an endless recursion of requests, a request handler is not allowed to fetch its own URL. It is still possible to cause an endless recursion with other means, so exercise caution if your app can be made to fetch requests for URLs supplied by the user.

You can set a deadline for a request, the most amount of time the service will wait for a response. By default, the deadline for a fetch is 5 seconds. The maximum deadline is 60 seconds for HTTP requests and 60 seconds for task queue and cron job requests.

Making requests to another App Engine app or Google service

If you are making requests to another App Engine app or Google service, you should consider telling the URL Fetch service to not follow redirects when invoking it.

Note: If you are making requests to another App Engine application, we recommend that you use its`appspot.com` domain name, rather than a custom domain for your app.

These settings do the following:

1.Optimizes the network path to the backend service, potentially leading to faster calls.

2.Adds an `X-Appengine-Inbound-Appid` header containing your app ID to the request, so the responding app can determine the source of incoming requests. For more information, see Asserting identity to other App Engine apps.

Secure connections and HTTPS

An app can fetch a URL with the HTTPS method to connect to secure servers. Request and response data are transmitted over the network in encrypted form.

In the PHP API, the proxy validates the host it is contacting by default, in order to detect "man in the middle" attacks between App Engine and the remote host when using HTTPS. This behaviour may be disabled by manually creating an SSL context and setting `verify_peer` to `false`.

Request headers

An app can set HTTP headers for the outgoing request.

When sending an HTTP POST request, if a `Content-Type` header is not set explicitly, the header is set to `x-www-form-urlencoded`. This is the content type used by web forms.

For security reasons, the following headers cannot be modified by the application:

- `Content-Length`
- `Host`
- `Vary`
- `Via`
- `X-Appengine-Inbound-Appid`
- `X-Forwarded-For`
- `X-ProxyUser-IP`

These headers are set to accurate values by App Engine, as appropriate. For example, App Engine calculates the`Content-Length` header from the request data and adds it to the request prior to sending.

Headers identifying request source

The following headers indicate the app ID of the requesting app:

- `User-Agent`. This header can be modified but App Engine will append an identifier string to allow servers to identify App Engine requests. The appended string has the format `"AppEngine-Google; (+http://code.google.com/appengine; appid: APPID)"`, where `APPID` is your app's identifier.

- `X-Appengine-Inbound-Appid`. This header cannot be modified, and is added automatically if the request is sent via the URL Fetch service when the follow redirects parameter is set to `False`.

Responses

The URL Fetch service returns all response data, including the response code, header and body.

By default, if the URL Fetch service receives a response with a redirect code, the service will follow the redirect. The service will follow up to 5 redirect responses, then return the final resource. You can use the API to tell the URL Fetch service to not follow redirects and just return a redirect response to the application.

By default, if the incoming response exceeds the maximum response size limit, the URL fetch service raises an exception. (See below for the amount of this limit.) You can tell the API to truncate the response instead of raising an exception.

URL Fetch and the development server

When your application is running in the development server on your computer, calls to the URL Fetch service are handled locally. The development server fetches URLs by contacting remote hosts directly from your computer, using whatever network configuration your computer is using to access the Internet.

When testing the features of your app that fetch URLs, be sure that your computer can access the remote hosts.

If your app is using the Google Secure Data Connector to access URLs on your intranet, be sure to test your app while connected to your intranet behind the firewall. Unlike App Engine, the development server does *not* use the SDC Agent to resolve intranet URLs. Only Google Apps and App Engine can authenticate with your SDC Agent.

Quotas and limits

Each URL Fetch request counts toward the URL Fetch API Calls quota.

Data sent in an HTTP or HTTPS request using the URL Fetch service counts toward the following quotas:

- Outgoing Bandwidth (billable)
- URL Fetch Data Sent

In addition to these quotas, data sent in an HTTPS request also counts toward the following quota:

- Secure Outgoing Bandwidth (billable)

Data received in response to an HTTP or HTTPS request using the URL Fetch service counts toward the following quotas:

- Incoming Bandwidth (billable)
- URL Fetch Data Received

In addition to these quotas, data received in response to an HTTPS request also counts toward the following quota:

- Secure Incoming Bandwidth (billable)

For more information on quotas, see Quotas, and the "Quota Details" section of the Admin Console.

In addition to quotas, the following limits apply to the use of the URL Fetch service:

Limit	Amount
request size	10 megabytes
response size	32 megabytes
maximum deadline (request handler)	60 seconds
maximum deadline (task queue and cron job handler)	60 seconds

Users PHP API Overview

App Engine applications can authenticate users using any one of three methods: Google Accounts, accounts on your own Google Apps domains, or OpenID identifiers. (Note that the support for OpenID is experimental.) An application can detect whether the current user has signed in, and can redirect the user to the appropriate sign-in page to sign in or, if your app uses Google Accounts authentication, create a new account. While a user is signed in to the application, the app can access the user's email address (or OpenID identifier if your app is using OpenID). The app can also detect whether the current user is an administrator, making it easy to implement admin-only areas of the app.

Note: Google no longer offers OpenID provider support. However, an app can still be an OpenID relying party. For apps that integrate with Google Apps Marketplace, we recommend using OAuth 2.0. Follow these instructions to migrate from OpenID to OAuth 2.0.

User authentication in PHP

The following example greets a user who has signed in to the app with a personalized message and a link to sign out. If the user is not signed in, the app offers a link to the sign-in page for Google Accounts, or to the page that requests an OpenID identifier.

```
use google\appengine\api\users\User;

use google\appengine\api\users\UserService;

$user = UserService::getCurrentUser();

if (isset($user)) {

  echo sprintf('Welcome, %s! (<a href="%s">sign out</a>)',

               $user->getNickname(),

               UserService::createLogoutUrl('/'));

} else {

  echo sprintf('<a href="%s">Sign in or register</a>',

               UserService::createLoginUrl('/'));

}
```

If your app uses OpenID and the user must sign in, your app will be redirected to the URL`/_ah/login_required`. You must create a page that lets the user sign in using an OpenID identifier. To specify this page, add an entry to your `app.yaml` file of the following form:

```
- url: /_ah/login_required

  script: do_openid_login.app
```

Enforcing sign in and admin access with app.yaml

If you have pages that require the user to be signed in in order to access, you can configure the handlers for those pages to require user sign-in with the `app.yaml` file. If a user accesses a URL configured to require sign-in and the user is not signed in, App Engine redirects the user to the appropriate sign-in page (when Google Accounts or Google Apps is used) or to `/_ah/login_required` (when OpenID is used), then directs the user back to your app's URL after signing in or registering successfully.

The handler configuration can also require that the user be a registered administrator for the application. This makes it easy to build administrator-only sections of the site, without having to implement a separate authorization mechanism.

To learn how to configure authentication for URLs, see Configuring a PHP App: Requiring Login or Administrator Status.

Authentication options

Your app can authenticate users using any one of 3 options:

- A Google Account
- An account on your Google Apps domain
- An OpenID identifier

About OpenID

OpenID is an open technology used for authenticating users across various web services. When a user creates an account on a service that acts as an *OpenID provider*, the user can then use a unique URL to that service as an OpenID identifier to sign in to any other service, known as an *OpenID relying party*, that allows OpenID sign-ins. For instance, if example.com is an OpenID provider, you can create an account on example.com, then use a URL given to you by that site that uniquely identifies you (such as `yourname.example.com` or`http://example.com/openid/yourname`) to sign in to sites that are OpenID relying parties.

If you set up your App Engine app to use OpenID for signing in, your app becomes an OpenID relying party. In other words, your app does not provide OpenID identifiers, but it requires them for sign in.

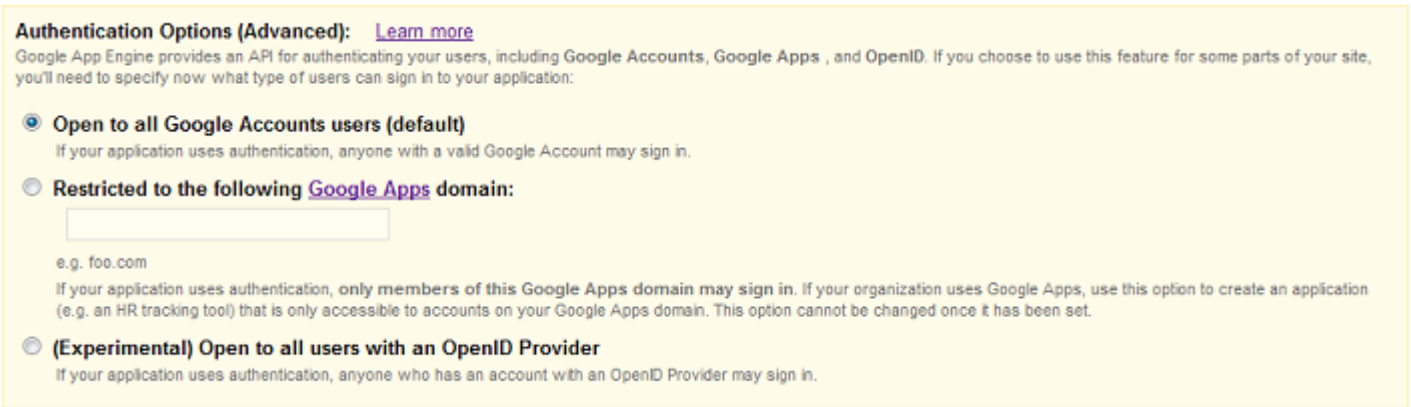
OpenID is one example of a type of authentication called *Federated Login*. Federated Login refers to any service that allows users to create a single ID or account and use it to authenticate with various services. OpenID is the only Federated Login supported by the Users service.

App Engine does not provide a user interface API for OpenID sign-in. Your OpenID sign-in user interface must allow the user to enter a URL that serves as an OpenID identifier. You might also include a pop-up menu listing the domain names of popular OpenID providers, along with a box for the user to type the unique part of the URL. For more information on the user interface for OpenID sign-in, see User Experience summary for Federated Login.

Note: App Engine does not support OpenID Attribute Exchange.

Choosing an authentication option

When you create your app, you choose the authentication option you want to use. By default, your app will use Google Accounts for authentication. To choose one of the other options, click Edit in the Authentication Options box. You'll then see the following:



Choose the option you want to use. Note that once you have created your app, your options for changing the authentication option are limited. In particular, you can only switch between Google Accounts and OpenID (Federated Login). If you do switch, user data in your datastore and user IDs are preserved and are still valid. Note that even if your app was written before OpenID support was added to App Engine, you can still switch to OpenID and access users' pre-existing data.

If you switch your app's authentication from Google Accounts to OpenID, your existing users can sign in to your app by using google.com as their OpenID provider.

To switch your app's authentication option from Google Accounts or OpenID, go to the Admin Console, click Application Settings, and choose the option you want from the Authentication Options menu. Be sure to click Save Settings.

Signing in and out

An application can detect whether a user has signed in to the app with your app's chosen authentication option. If the user is not signed in, the app can direct the user to Google Accounts to sign in or create a new Google account, or to an OpenID sign-in page. The app gets the URL for the sign-in page by calling a method of the Users API. The app can display this URL as a link, or it can issue an HTTP redirect to the URL when the user visits a page that requires authentication.

If your app uses Google Accounts or Google Apps for authentication, the name of your application appears on the sign-in page when the user signs in to your application, using the application name you chose when registering the application. You can change your application name in the "Application Settings" section of the Admin Console.

If your app uses OpenID, the sign-in page displays the hostname instead of the app name.

Once the user has signed in or created a Google account, the user is redirected back to your application. The app provides the redirect URL to the method that generates the sign-in URL.

The Users API includes a method to generate a URL for signing out of the app. The sign-out URL de-authenticates the user from the app, then redirects back to the app's URL without displaying anything.

A user is not signed in to an application until she is prompted to do so by the app and enters her account's email address and password, or OpenID identifier. This is true even if the user has signed in to other applications using her Google Account.

Accessing account information

While a user is signed in to an app, the app can access the account's email address or OpenID identifier for every request the user makes to the app. The app can also access a user ID that identifies the user uniquely, even if the user changes the email address for her account.

The app can also determine whether the current user is an administrator (a "developer") for the app. You can use this feature to build administrative features for the app, even if you don't authenticate other users. The Go, Java, PHP and Python APIs make it easy to configure URLs as "administrator only".

Note: Every user has the same user ID for all App Engine applications. If your app uses the user ID in public data, such as by including it in a URL parameter, you should use a hash algorithm with a "salt" value added to obscure the ID. Exposing raw IDs could allow someone to associate a user's activity in one app with that in another, or get the user's email address by coercing the user to sign in to another app.

Google accounts and the development server

The development server simulates the Google Accounts system using a dummy sign-in screen. When your application calls the Users API to get the URL for the sign-in screen, the API returns a special development server URL that prompts for an email address, but no password. You can type any email address into this prompt, and the app will behave as if you are signed in with an account with that address.

The dummy sign-in screen also includes a checkbox that indicates whether the dummy account is an administrator. If you check this box, the app will behave as if you are signed in using an administrator account.

Similarly, the Users API returns a sign-out URL that cancels the dummy sign-in.

The unique ID for a User object in the development server is calculated from the email address. Two unique email addresses always represent two unique users in the development server.

User Objects

An instance of the User class represents a user. User instances are unique and comparable. If two instances are equal, then they represent the same user.

The application can access the User instance for the current user by calling theUserService::getCurrentUser() function.

```
use google\appengine\api\users\User;

use google\appengine\api\users\UserService;

$user = UserService::getCurrentUser();

if (!isset($user)) {

  // The user is not logged in

} else {

  echo 'Hello, ' . $user->getNickname();

}
```

You can use the UserService::getCurrentUser() function no matter which authentication option your app uses.

A User instance can be also constructed from an email address:

```
$user = new User("Albert.Johnson@example.com");
```

Or, if you have a federated_identity, you can use it to create a User instance:

```
$user = new User(null, "http://example.com/id/ajohnson");
```

If the User::__construct() is called with an email address that does not correspond with a valid Google account, the object will be created but it will not correspond with a real Google account. This will be the case even if someone creates a Google account with the given email address after the object is stored. A User value with an email address that does not represent a Google account at the time it is created will never match a User value that represents a real user.

The User object for a valid user can provide a unique ID value for the user that stays the same even if the user changes her email address. The `getUserId()` method returns this ID, a `string` value.

The User object has the same form no matter which method of authentication your app uses. If you switch authentication options from Google Accounts to OpenID, existing User objects in the datastore are still valid.

Login URLs

The Users API provides functions for constructing URLs that allow the user to sign in or sign out, then be redirected back to your application.

`UserService::createLoginUrl()` and `UserService::createLogoutUrl()` each take a destination URL for the application, and return a URL for signing in or signing out that redirects back to the given URL afterward.

`UserService::createLoginUrl()` also takes an optional OpenID identifier argument,`federated_identity`, which you can use if your app is set to support OpenID. By default, the`federated_identity` argument uses Google's OpenID provider service.

```
use google\appengine\api\users\User;

use google\appengine\api\users\UserService;

$user = UserService::getCurrentUser();

if (isset($user)) {

  echo sprintf('Welcome, %s! (<a href="%s">sign out</a>)',

               $user->getNickname(),

               UserService::createLogoutUrl('/'));

} else {

  echo sprintf('<a href="%s">Sign in or register</a>',

               UserService::createLoginUrl('/'));

}
```

The development web server simulates Google Accounts using its own sign-in and sign-out facilities. When you sign in to your application on the development web server, the server prompts you for an email address to use for the session. See The Development Web Server for more information.

Tip: An easy way to restrict access to a part of your application to signed in users is to use the `login: required` configuration element for the URL handler. See Configuring an App.

Admin Users

An application can test whether the currently signed-in user is a registered administrator for the application. An administrator is a user who can access the Administration Console for the application. You can use the Administration Console to manage which users have administrator status.

The function UserService::isCurrentUserAdmin() returns `True` if the current user is an administrator for the application.

```
use google\appengine\api\users\User;

use google\appengine\api\users\UserService;

$user = UserService::getCurrentUser();

if (isset($user)) {

  echo 'Welcome, ' . $user->getNickname();

  if (UserService::isCurrentUserAdmin()) {

    echo 'Go to admin area';

  }

}
```

Tip: An easy way to restrict access to a part of your application to administrators is to use the `login: admin` configuration element for the URL handler. See Configuring an App.

Task Queue PHP API Overview

With the Task Queue API, applications can perform work outside of a user request, initiated by a user request. If an app needs to execute some background work, it can use the Task Queue API to organize that work into small, discrete units, called tasks. The app adds tasks to task queues to be executed later.

Warning: There is currently no support for pull queues or the REST API in the PHP runtime for Google App Engine.

Task Queue concepts

Tasks queues are an efficient and powerful tool for background processing; they allow your application to define tasks, add them to a queue, and then use the queue to process them in aggregate. You name queues and configure their properties in a configuration file named `queue.yaml`. A queue can be one of two types—*push* or *pull*—based on your needs.

Push queues function only within the App Engine environment. These queues are the best choice for applications whose tasks work only with App Engine tools and services. With push queues, you simply configure a queue and add tasks to it. App Engine handles the rest. Push queues are easier to implement, but are restricted to use within App Engine. For more information about push queues and examples of how to use them, see Using Push Queues.

In summary, push queues allow you to process tasks within App Engine at a steady rate and App Engine scales computing resources according to the number of tasks in your queue.

The default queue

For convenience, App Engine provides a default push queue for each application (there is no default pull queue). If you do not name a queue for a task, App Engine automatically inserts it into the default queue. You can use this queue immediately without any additional configuration. All modules and versions of the same application share the same default task queue.

The default queue is preconfigured with a throughput rate of 5 task invocations per second. If you want to change the preconfigured settings, simply define a queue named `default` in `queue.yaml`. Code may always insert new tasks into the default queue, but if you wish to disable execution of these tasks, you may do so by clicking the Pause Queue button in the Task Queues tab of the Administration Console.

Named queues

While the default queue makes it easy to enqueue tasks with no configuration, you can also create custom queues by defining them in `queue.yaml`. Custom queues allow you to more effectively handle task processing by grouping similar types of tasks. You can control the processing rate—and a number of other settings—based specifically on the type of task in each queue.

All versions of an application share the same named task queues.

For more information about configuring queues in `queue.yaml`, please see Task Queue Configuration.

Task queues in the Administration Console

You can manage task queues for an application using the Task Queue tab of the Administration Console. The Task Queue tab lists all of the queues in the application. Clicking on a queue name brings up the Task Queue Details page where you can see all of the tasks scheduled to run in a queue and you can manually delete individual tasks or purge every task from a queue. This is useful if a task in a push queue cannot be completed successfully and is stuck waiting to be retried. You can also pause and resume a queue on this page.

You can view details of individual tasks by clicking the task name from the list of tasks on the Task Queue Detailspage. This page allows you to debug why a task did not run successfully. You can also see information about the previous run of the task as well as the task's body.

Important: The default queue appears in the Console only after the app has enqueued a task to it for the first time.

Task concepts

A *task* is a unit of work to be performed by the application. Each task is an object of the `PushTask` class. Each Task object contains an endpoint (with a request handler for the task and an optional data payload that parameterizes the task). You can enqueue push tasks to a queue defined in `queue.yaml`.

Task names

You may optionally assign a name to a task. Task names must be unique within a queue. Once a named task is added to a queue, any subsequent attempt to insert another task with the same name into the same queue will fail.

Note that task names do not provide an absolute guarantee of once-only semantics. In rare cases, multiple calls to create a task of the same name may succeed. It's also possible in exceptional cases for a task to run more than once—even if it was only created once.

Task names may be up to 500 characters long, and must be a combination of one or more digits, letters a–z, underscores, and/or dashes, satisfying the following regular expression:

```
[0-9a-zA-Z\-\_]+
```

If a push task is created successfully, it will eventually be deleted (at most seven days after the task successfully executes). Once deleted, its name can be reused.

If a pull task is created successfully, your application needs to delete the task after processing. The system may take up to seven days to recognize that a task has been deleted; during this time, the task name remains unavailable. Attempting to create another task during this time with the same name will result in an "item exists" error. The system offers no method to determine if deleted task names are still in the system. To avoid these issues, we recommend that you let App Engine generate the task name automatically.

Using Push Queues in PHP

In App Engine push queues, a *task* is a unit of work to be performed by the application.

Using push queues

A PHP app sets up queues using a configuration file named `queue.yaml` (see PHP Task Queue Configuration). If an app does not have a `queue.yaml` file, it has a queue named `default` with some default settings.

To enqueue a task, create a PushTask object and call its `add()` method. You can add to a queue specified in`queue.yaml` by supplying a queue name argument to `add()`. Alternatively, calling `add()` with no arguments will add the task to the default queue.

The following code creates a task that will be sent as a POST request to the `/worker` handler of the application. The task contains name and action data, and will be processed by the default queue:

```
use google\appengine\api\taskqueue\PushTask;

$task = new PushTask('/worker.php', ['name' => 'john doe', 'action' =>
'send_reminder']);

$task_name = $task->add();
```

You can also add tasks in bulk to a queue using PushQueue. In the following example, two PushTask objects are added to a PushQueue using the `addTasks()` method.

```
use google\appengine\api\taskqueue\PushTask;

use google\appengine\api\taskqueue\PushQueue;

$task1 = new PushTask('/someUrl');

$task2 = new PushTask('/someOtherUrl');

$queue = new PushQueue();

$queue->addTasks([$task1, $task2]);
```

Push task execution

App Engine executes push tasks by sending HTTP requests to your app. Specifying a programmatic asynchronous callback as an HTTP request is sometimes called a web hook. The web hook model enables efficient parallel processing.

The task's URL determines the *handler* for the task and the *module* that runs the handler.

The handler is determined by the path part of the URL (the forward-slash separated string following the hostname), which is specified as the first argument (the `url_path`) of the PushTask constructor. The path must be relative and local to your application's root directory.

The module and version in which the handler runs is determined by:

- The "Host" header that you can add to the `$options` when you create a `PushTask`.
- The `target` directive in the `queue.yaml` file.

If you do not specify any of these parameters, the task will run in the same module/version in which it was enqueued, subject to these rules:

- If the default version of the app enqueues a task, the task will run on the default version. Note that if the app enqueues a task and the default version is changed before the task actually runs, the task will be executed in the new default version.
- If a non-default version enqueues a task, the task will always run on that same version.

Note: If you are using modules along with a dispatch file, a task's URL may be intercepted and re-routed to another module.

Task request headers

Requests from the Task Queue service contain the following HTTP headers:

- `X-AppEngine-QueueName`, the name of the queue (possibly `default`)
- `X-AppEngine-TaskName`, the name of the task, or a system-generated unique ID if no name was specified
- `X-AppEngine-TaskRetryCount`, the number of times this task has been retried; for the first attempt, this value is 0. This number includes attempts where the task failed due to a lack of available instances and never reached the execution phase.
- `X-AppEngine-TaskExecutionCount`, the number of times this task has previously failed during the execution phase. This number does not include failures due to a lack of available instances.
- `X-AppEngine-TaskETA`, the target execution time of the task, specified in milliseconds since January 1st 1970.

These headers are set internally by Google App Engine. If your request handler finds any of these headers, it can trust that the request is a Task Queue request. If any of the above headers are present in an external user request to your app, they are stripped. The exception being requests from logged in administrators of the application, who are allowed to set the headers for testing purposes.

Google App Engine issues Task Queue requests from the IP address `0.1.0.2`.

Task deadlines

A task must finish executing and send an HTTP response value between 200–299 within a time interval that depends on the scaling type of the App Engine module that receives the request. This deadline is separate from user requests, which have a 60-second deadline.

Tasks targeted at an automatic scaled module must finish execution within 10 minutes. If you have tasks that require more time or computing resources, they can be sent to manual or basic scaling modules, where they can run up to 24 hours. If the task fails to respond within the deadline, or returns an invalid response value, the task will be retried as described in the next section.

Task retries

If a push task request handler returns an HTTP status code within the range 200–299, App Engine considers the task to have completed successfully. If the task returns a status code outside of this range, App Engine retries the task until it succeeds. The system backs off gradually to avoid flooding your application with too many requests, but schedules retry attempts for failed tasks to recur at a maximum of once per hour.

You can also configure your own scheme for task retries using the retry_parameters directive in `queue.yaml`.

When implementing the code for tasks (as worker URLs within your app), it is important to consider whether the task is idempotent. App Engine's Task Queue API is designed to only invoke a given task once; however, it is possible in exceptional circumstances that a task may execute multiple times (such as in the unlikely case of major system failure). Thus, your code must ensure that there are no harmful side-effects of repeated execution.

The rate of task execution

You set the maximum processing rate for the entire queue when you configure the queue. App Engine uses atoken bucket algorithm to execute tasks once they've been delivered to the queue. Each queue has a token bucket, and each bucket holds a certain number of

tokens. Your app consumes a token each time it executes a task. If the bucket runs out of tokens, the system pauses until the bucket has more tokens. The rate at which the bucket is refilled is the limiting factor that determines the rate of the queue. See Defining Push Queues and Processing Rates for more details.

To ensure that the Task Queue system does not overwhelm your application, it may throttle the rate at which requests are sent. This throttled rate is known as the *enforced rate*. The enforced rate may be decreased when your application returns a 503 HTTP response code, or if there are no instances able to execute a request for an extended period of time. You can view the enforced rate on the Task Queue tab of the Administration Console.

The order of task execution

The order in which tasks are executed depends on several factors:

- The position of the task in the queue. App Engine attempts to process tasks based on FIFO (first in, first out) order. In general, tasks are inserted into the end of a queue, and executed from the head of the queue.

- The backlog of tasks in the queue. The system attempts to deliver the lowest latency possible for any given task via specially optimized notifications to the scheduler. Thus, in the case that a queue has a large backlog of tasks, the system's scheduling may "jump" new tasks to the head of the queue.

- The value of the task's delay_seconds property. This specifies the minimum number of seconds to wait before executing a task.

URL endpoints

Push tasks reference their implementation via URL. For example, a task which fetches and parses an RSS feed might use a worker URL called `/app_worker/fetch_feed`. You must specify this worker URL when creating a PushTask. In general, you can use any URL as the worker for a task, so long as it is within your application; all task worker URLs must be specified as relative URLs:

```
use google\appengine\api\taskqueue\PushTask;

(new PushTask('/path/to/my/worker', ['data_for_task' => 1234]))->add();
```

Securing URLs for tasks

You can prevent users from accessing URLs of tasks by restricting access to administrator accounts. Task queues can access admin-only URLs. You can restrict a URL by adding `login: admin` to the handler configuration in `app.yaml`.

An example might look like this in `app.yaml`:

```
application: hello-tasks

version: 1

runtime: php

api_version: 1

threadsafe: true

handlers:

- url: /tasks/process

  script: process.php

  login: admin
```

Note: While task queues can use URL paths restricted with `login: admin`, they *cannot* use URL paths restricted with `login: required`.

For more information see PHP Application Configuration: Requiring Login or Administrator Status.

To test a task web hook, sign in as an administrator and visit the URL of the handler in your browser.

Push queues and the development server

When your app is running in the development server, tasks are automatically executed at the appropriate time just as in production.

To disable tasks from running in the development server, run the following command:

```
dev_appserver.py --disable_task_running
```

You can examine and manipulate tasks from the developer console at:`http://localhost:8000/_ah/admin/taskqueue`.

To execute tasks, select the queue by clicking on its name, select the tasks to execute, and click Run Now. To clear a queue without executing any tasks, click Purge Queue.

The development server and the production server behave differently:

- The development server doesn't respect the `<rate>` and `<bucket-size>` attributes of your queues. As a result, tasks are executed as close to their ETA as possible. Setting a rate of 0 doesn't prevent tasks from being executed automatically.
- The development server doesn't retry tasks.
- The development server doesn't preserve queue state across server restarts.

Quotas and limits for push queues

Enqueuing a task in a push queue counts toward the following quotas:

- Task Queue Stored Task Count
- Task Queue Stored Task Bytes
- Task Queue API Calls

The Task Queue Stored Task Bytes quota is configurable in This quota counts towards your Stored Data (billable) quota.

Execution of a task counts toward the following quotas:

- Requests
- Incoming Bandwidth
- Outgoing Bandwidth

The act of executing a task consumes bandwidth-related quotas for the request and response data, just as if the request handler were called by a remote client. When the task queue processes a task, the response data is discarded.

Once a task has been executed or deleted, the storage used by that task is reclaimed. The reclaiming of storage quota for tasks happens at regular intervals, and this may not be reflected in the storage quota immediately after the task is deleted.

For more information on quotas, see Quotas, and the "Quota Details" section of the Admin Console.

The following limits apply to the use of push queues:

Push Queue Limits	
Maximum task size	100KB
Maximum number of active queues (not including the default queue)	Free apps: 10 queues, Billed apps: 100 queues
Queue execution rate	500 task invocations per second per queue
Maximum countdown/ETA for a task	30 days from the current date and time
Maximum number of tasks that can be added in a batch	100 tasks
Maximum number of tasks that can be added in a transaction	5 tasks