HTTP request and response

Accessing and manipulating Zope’s HTTP request and response objects programmatically.

Introduction

This chapter explains the basics of Zope HTTP requests and responses:

* request and response objects lifecycle;
* data which can be extracted from the request;
* data which can be placed on the response.

Lifecycle

Unlike some other web frameworks, in Plone you do not explicitly create or return HTTP response objects. A HTTP request object always has a HTTP response object associated with it, and the response object is created as soon as the request hits the webserver.

The response is available for the whole lifetime of request processing. This effectively allows you to set and modify response headers at any point in the code.

Webservers

Usually Plone runs on Zope’s [ZServer](https://github.com/zopefoundation/ZServer/blob/master/src/ZServer/README.txt) (based on Sam Rushing’s [Medusa](https://web.archive.org/web/20110714080523/http:/www.amk.ca/python/code/medusa.html)). Other alternatives are [WSGI](http://ivory.idyll.org/articles/wsgi-intro/what-is-wsgi.html) compatible web servers like [Repoze](http://repoze.org/).

The web server will affect how your HTTP objects are constructed.

HTTP Request

All incoming HTTP requests are wrapped in Zope’s [ZPublisher](http://www.python.org/) [HTTPRequest](https://github.com/zopefoundation/Zope/blob/master/src/ZPublisher/HTTPRequest.py) objects. This is a multi-mapping: it contains mappings for environment variables, other variables, form data, and cookies, but the keys of all these mappings can also be looked up directly on the request object (i.e. request['some\_form\_id'] and request.form['some\_form\_id'] are equivalent).

Usually your view function or instance will receive an HTTP request object, along with a traversed context, as its construction parameter.

You can access the request in your view:

**from** **Products.Five.browser** **import** BrowserView

**class** **SampleView**(BrowserView):

**def** \_\_init\_\_(self, context, request):

*# Each view instance receives context and request as construction parameters*

self.context = context

self.request = request

**def** \_\_call\_\_(self):

*# Entry point of request processing*

*# Dump out incoming request variables*

print self.request.items()

Request method

The request method (GET or POST) can be read:

request["REQUEST\_METHOD"] == "POST" *# or "GET"*

Request URL

To get the requested URL:

**>>>** request["ACTUAL\_URL"]

'http://localhost:8080/site'

To get the URL of the served object use the following (this might be different from the requested URL, since Plone does all kinds of default page and default view magic):

**>>>** request["URL"]

'http://m.localhost:8080/site/matkailijallefolder/@@frontpage'

URLs, as accessed above, do not contain query string.

Query string

The unparsed query string can be accessed.

E.g. if you go to http://localhost:8080/site?something=foobar:

**>>>** self.request["QUERY\_STRING"]

'something=foobar'

If the query string is not present in the HTTP request, it is an empty string.

You can also use the request.form dictionary to access parsed query string content.

Request path

The request URI path can be read from request.path, which returns a list of path components. request.path is a virtual path, and has the site id component removed from it.

Example:

reconstructed\_path = "/".join(request.path)

Other possible headers:

('PATH\_INFO', '/plonecommunity/Members')

('PATH\_TRANSLATED', '/plonecommunity/Members')

What’s the difference?

REQUEST\_URI

To get the variable which corresponds to REQUEST\_URI in e.g. PHP the following helps:

*# Concatenate the user-visible URL and query parameters*

full\_url = request.ACTUAL\_URL + "?" + request.QUERY\_STRING

parsed = urlparse.urlsplit(full\_url)

*# Extract path part and add the query if it existed*

uri = parsed[2]

**if** parsed[3]:

uri += "?" + parsed[3]

For more information, see:

* <http://www.teamrubber.com/blog/_serverrequest_uri-in-zope/>
* <http://www.doughellmann.com/PyMOTW/urlparse/index.html>

Request client IP

Example:

**def** get\_ip(request):

*""" Extract the client IP address from the HTTP request in a proxy-compatible way.*

*@return: IP address as a string or None if not available*

*"""*

**if** "HTTP\_X\_FORWARDED\_FOR" **in** request.environ:

*# Virtual host*

ip = request.environ["HTTP\_X\_FORWARDED\_FOR"]

**elif** "HTTP\_HOST" **in** request.environ:

*# Non-virtualhost*

ip = request.environ["REMOTE\_ADDR"]

**else**:

*# Unit test code?*

ip = **None**

**return** ip

For functional tests based on zope.testbrowser use the addHeader method to add custom headers to a browser.

GET variables

HTTP GET variables are available in request.form if the REQUEST\_METHOD was GET.

Example:

*# http://yoursite.com/@@testview/?my\_param\_id=something*

print self.request.form["my\_param\_id"]

POST variables

HTTP POST varibles are available in request.form:

print request.form.items() *# Everything POST brought to us*

There is no difference in accessing GET and POST variables.

Request body

The request body can be retrieved from the [HTTPRequest](https://github.com/zopefoundation/Zope/blob/master/src/ZPublisher/HTTPRequest.py) object by using the get method with the key BODY:

print request.get('BODY') *# Prints the content of the request body*

HTTP headers

HTTP headers are available through request.get\_header() and the request.environ dictionary.

Example:

referer = self.request.get\_header("referer") *# Page referer (the page from user came from)*

**if** referer == **None**: *# referer will be none if it was missing*

**pass**

Dumping all headers:

**for** name, value **in** request.environ.items():

print "*%s*: *%s*" % (name, value)

A Management Interface Python script to dump all HTTP request headers:

**from** **StringIO** **import** StringIO

request = container.REQUEST

response = request.response

buffer = StringIO()

response.setHeader("Content-type", "text/plain")

**for** name, value **in** request.environ.items():

print >> buffer, "*%s*: *%s*" % (name, value)

**return** buffer.getvalue()

Query string

To access the raw HTTP GET query string:

query\_string = request["QUERY\_STRING"]

Web environment

The web server exposes its own environment variables in request.other ([ZServer](https://github.com/zopefoundation/ZServer/blob/master/src/ZServer/README.txt)) or request.environ ([Repoze](http://repoze.org/) and other [WSGI](http://ivory.idyll.org/articles/wsgi-intro/what-is-wsgi.html)-based web servers):

print request.other.items()

user\_agent = request.other["HTTP\_USER\_AGENT"]

user\_agent = request.environ["HTTP\_USER\_AGENT"] *# WSGI or Repoze server*

Hostname

Below is an example to get the HTTP server name in a safe manner, taking virtual hosting into account:

**def** get\_hostname(request):

*""" Extract hostname in virtual-host-safe manner*

*@param request: HTTPRequest object, assumed contains environ dictionary*

*@return: Host DNS name, as requested by client. Lowercased, no port part.*

*Return None if host name is not present in HTTP request headers*

*(e.g. unit testing).*

*"""*

**if** "HTTP\_X\_FORWARDED\_HOST" **in** request.environ:

*# Virtual host*

host = request.environ["HTTP\_X\_FORWARDED\_HOST"]

**elif** "HTTP\_HOST" **in** request.environ:

*# Direct client request*

host = request.environ["HTTP\_HOST"]

**else**:

**return** **None**

*# separate to domain name and port sections*

host=host.split(":")[0].lower()

**return** host

See also

* <http://httpd.apache.org/docs/2.1/mod/mod_proxy.html#x-headers>
* <http://zotonic.googlecode.com/hg/doc/varnish.zotonic.vcl> (X-Forwarded-Host)

Request port

It is possible to extract the Zope instance port from the request. This is useful e.g. for debugging purposes if you have multiple ZEO front ends running, and you want to identify them:

port = request.get("SERVER\_PORT", **None**)

The SERVER\_PORT variable returns the port number as a string, not an integer.

This port number is not the one visible to the external traffic (port 80, HTTP)

Published object

request["PUBLISHED"] points to a view, method or template which was the last item in the traversing chain to be called to render the actual page.

To extract the relevant content item from this information you can do e.g. in the after publication hook:

**def** find\_context(request):

*"""Find the context from the request*

*http://stackoverflow.com/questions/10489544/getting-published-content-item-out-of-requestpublished-in-plone*

*"""*

published = request.get('PUBLISHED', **None**)

context = getattr(published, '\_\_parent\_\_', **None**)

**if** context **is** **None**:

context = request.PARENTS[0]

**return** context

* You might also want to filter out CSS etc. requests
* Please note that request[PUBLISHED] is set after language negotiation and authentication
* [More complete example](https://github.com/miohtama/silvuple/blob/master/silvuple/negotiator.py)

Flat access

GET, POST and web environment variables are flat mapped to the request object as a dictionary look up:

*# Comes from POST*

request["input\_username"] == request.form["input\_username"]

*# Comes from environ*

request.get('HTTP\_USER\_AGENT') == request.environ["HTTP\_USER\_AGENT"]

Request mutability

Even if you can write and add your own attributes to HTTP request objects, this behavior is discouraged. If you need to create cache variables for request lifecycle use [annotations](https://pypi.python.org/pypi/zope.annotation/3.4.1).

Add link to internal annotations examples when written.

Accessing HTTP request outside context

There are often cases where you would like to get hold of the HTTP request object, but the underlying framework does not pass it to you. In these cases you have two ways to access the request object:

* Use *acquisition* to get the request object from the site root. When Plone site traversal starts, the HTTP request is assigned to current site object as the site.REQUEST attribute.
* Use <https://pypi.python.org/pypi/five.globalrequest>.

Example of getting the request using acquisition:

*# context is any traversed Plone content item*

request = getattr(context, "REQUEST", **None**)

**if** request **is** **not** **None**:

*# Do the normal flow*

...

**else**:

*# This code path was not initiated by an incoming web server request*

*# Handle cases like*

*# - command line scripts*

*# - add-on installer*

*# - code called during Zope start up*

*# -etc.*

...

zope.globalrequest.getRequest

See

* <https://pypi.python.org/pypi/five.globalrequest>

HTTP response

Usually you do not return HTTP responses directly from your views. Instead, you modify the existing HTTP response object (associated with the request) and return the object which will be HTTP response payload.

The returned payload object can be:

* a string (str) 8-bit raw data; or
* an iterable: the response is streamed, instead of memory-buffered.

Accessing response

You can access the HTTP response if you know the request:

**from** **Products.Five.browser** **import** BrowserView

**class** **SampleView**(BrowserView):

**def** \_\_init\_\_(context, request):

*# Each view instance receives context and request as construction parameters*

self.context = context

self.request = request

**def** \_\_call\_\_(self):

response = self.request.response

**return** "<html><body>Hello world!</body></html>"

Response headers

Use [HTTPResponse](https://github.com/zopefoundation/Zope/blob/master/src/ZPublisher/HTTPResponse.py) setHeader() to set headers:

*# The response is a dynamically generated image*

self.request.response.setHeader("Content-type", "image/jpeg")

**return** image\_data

Content disposition

The Content-Disposition header is used to set the filename of a download. It is also used by Flash 10 to check whether Flash download is valid.

Example of setting the download and downloadable filename:

response = self.request.response

response.setHeader("Content-type", "text/x-vCard; charset=utf-8")

response.setHeader("Content-Transfer-Encoding", "8bit")

cd = 'attachment; filename=*%s*.vcf' % (context.id)

response.setHeader('Content-Disposition', cd)

For more information, see:

* <http://www.littled.net/new/2008/10/17/plone-and-flash-player-10/>
* <http://support.microsoft.com/kb/260519>

Return code

Use HTTPResponse.setStatus(self, status, reason=None, lock=None) to set HTTP return status (“404 Not Found”, “500 Internal Error”, etc.).

If lock=True, no further modification of the HTTPResponse status are allowed, and will fail silently.

Response body

You might want to read or manipulate the response body in the post-publication hook.

The response body is not always a string or basestring: it can be a generator or iterable for blob data.

The body is available as the response.body attribute.

You can change the body using setBody and locking it:

*#lets empty the body and lock it*

response.setBody('', lock=**True**)

If lock=True, no further modification of the HTTPResponse body are allowed, and will fail silently.

Redirects

**Real redirects**

Use the response.redirect() method:

*# This will send a "302 Temporary Redirect" notification to the browser*

response.redirect(new\_url)

*# This will send a "301 Permanent Redirect" notification to the browser*

response.redirect(new\_url, status=301)

You can lock the status to not let other change the status later in the process

response.redirect(new\_url, lock=**True**)

**JavaScript redirects**

You can invoke this JavaScript redirect trick from a page template head slot in a hacky way

Cookies

See [cookies documentation](https://docs.plone.org/develop/plone/sessions/cookies.html).

Middleware-like hooks

Plone does not have a middleware concept, as everything happens through traversal. Middleware behavior can be emulated with the *before traverse* hook. This hook can be installed on any persistent object in the traversing graph. The hook is persistent, so it is a database change and must be installed using custom GenericSetup Python code.

Before traverse hooks cannot create new HTTP responses, or return alternative HTTP responses. Only exception-like HTTP response modification is supported, e.g. HTTP redirects. If you need to rewrite the whole response, the post-publication hook must be used.

For more information, see:

* <http://blog.fourdigits.nl/changing-your-plone-theme-skin-based-on-the-objects-portal_type>
* <http://zebert.blogspot.com/2008_01_01_archive.html>
* <http://svn.repoze.org/thirdparty/zopelib/branches/2.9.8/ZPublisher/tests/testBeforeTraverse.py>

Examples:

* Redirector: <https://plonegomobile.googlecode.com/svn/trunk/gomobile/gomobile.mobile/gomobile/mobile/postpublication.py>

Transform chain

Transform chain is a hook into repoze.zope2 that allows third party packages to register a sequence of hooks that will be allowed to modify the response before it is returned to the browser.

It is used e.g. by plone.app.caching.

More information

* <https://pypi.python.org/pypi/plone.transformchain>

Post-publication hook

The post-publication hook is run when:

* the context object has been traversed;
* after the view has been called and the view has rendered the response;
* before the response is sent to the browser;
* before the transaction is committed.

This is practical for caching purposes: it is the ideal place to determine and insert caching headers into the response.

Read more at the [plone.postpublicationhook package page](https://pypi.python.org/pypi/plone.postpublicationhook/).

Custom redirect mappings

Below is an example how to install an event handler which checks in the site root for a TTW Python script and if such exist it asks it to provide a HTTP redirect.

This behavior allows you to write site-wide redirects

* In Python (thank god no Apache regular expressions)
* Redirects can access Plone content items
* You can have some redirects migrated from the old (non-Plone) sites

Add the event subscriber to configure.zcml:

**<configure**

xmlns="http://namespaces.zope.org/zope"

xmlns:browser="http://namespaces.zope.org/browser"

xmlns:plone="http://namespaces.plone.org/plone"

i18n\_domain="example.dexterityforms"**>**

...

**<subscriber**

for="zope.traversing.interfaces.IBeforeTraverseEvent"

handler=".redirect.check\_redirect"

**/>**

**</configure>**

Create a file redirect.py and add the code below. Remember to add url to *Parameter list* of the script on the script edit view:

*"""*

*Call a custom TTW script and allow it to handle redirects.*

*Use the Management Interface to add a ``Script (Python)`` item named ``redirect\_handler``*

*to your site root - you can edit this script in fly to change the redirects.*

*\* Redirect script must contain ``url`` in its parameter list*

*"""*

**import** **logging**

**from** **zope.component.hooks** **import** getSite

**from** **zExceptions** **import** Redirect

**from** **Products.CMFCore.interfaces** **import** ISiteRoot

logger = logging.getLogger("redirect")

**def** check\_redirect(event):

*"""*

*Check if we have a custom redirect script in Zope application server root.*

*If we do then call it and see if we get a redirect.*

*The script itself is TTW Python script which may return*

*string in the case of redirect or None if no redirect is needed.*

*For more examples, check*

*https://github.com/zopefoundation/Zope/blob/master/src/Zope2/App/tests/testExceptionHook.py*

*"""*

site = getSite()

request = event.request

url = request["ACTUAL\_URL"]

**if** "no\_redirect" **in** request.form:

*# Use no\_redirect query parameter to disable this behavior in the case*

*# you mess up with the redirect script*

**return**

*# Check if we have a redirect handler script in the site root*

**if** "redirect\_handler" **in** site:

**try**:

*# Call the script and get its output*

value = site.redirect\_handler(url)

**except** Exception, e:

*# No silent exceptions plz*

logger.error("Redirect exception for URL:" + url)

logger.exception(e)

**return**

**if** value **is** **not** **None** **and** value.startswith("http"):

*# Trigger redirect, but only if the output value looks sane*

**raise** Redirect, value

Then an example redirect\_handler script added through the Management Interface. Remember to add url to the *Parameter List* field of TTW (through the web) interface:

**if** "blaablaa" **in** url:

**return** "http://webandmobile.mfabrik.com"

Or more complex example:

*# Don't leak non-themed interface fom port 80*

**if** ("manage.") **in** url **and** (**not** "8080" **in** url):

**return** "http://manage.underconstruction.mfabrik.com:8080/LS"

**if** url == "http://underconstruction.mfabrik.com/":

**return** "http://underconstruction.mfabrik.com/special-front-page"

*# Redirect to the actual front page*

**if** url == "http://site.com/":

**return** "http://www.site.com/special-front-page"

**if** url == "http://www.site.com/":

**return** "http://www.site.com/special-front-page"

**if** url.startswith("http://underconstruction.mfabrik.com/"):

**return** url.replace("underconstruction.mfabrik.com", "www.site.com")

*# Make sure that search engines and visitors access the site only using www. prefix*

**if** url.startswith("http://site.com/"):

**return** url.replace("site.com", "www.site.com")

Extracting useful information in the post-publication hook

Example:

**from** **zope.component** **import** adapter, getUtility, getMultiAdapter

**from** **plone.postpublicationhook.interfaces** **import** IAfterPublicationEvent

**from** **Products.CMFCore.interfaces** **import** IContentish

**def** get\_contentish(object):

*"""*

*Traverse acquisition upwards until we get contentish object used for the HTTP response.*

*"""*

contentish = object

**while** **not** IContentish.providedBy(contentish):

**if** hasattr(contentish, "aq\_parent"):

contentish = contentish.aq\_parent

**else**:

**break**

**return** contentish

*# This must be referred in configure.zcml*

**@adapter**(Interface, IAfterPublicationEvent)

**def** language\_fixer(object, event):

*""" Redirect mobile users to mobile site using gomobile.mobile.interfaces.IRedirector.*

*Note: Plone does not provide a good hook doing this before traversing, so we must*

*do it in post-publication. This adds extra latency, but is doable.*

*"""*

request = event.request

response = request.response

*# object can be a page template, view, whichever happens to be at the very end of traversed acquisition chain*

context = get\_contentish(object)

Cross-origin resource sharing (CORS)

Complete.

* <http://enable-cors.org/>
* <https://developer.mozilla.org/En/HTTP_access_control>
* [**Documentation**](https://docs.plone.org/index.html)

* [**Developing for Plone**](https://docs.plone.org/develop/index.html)

* [**Programming with Plone**](https://docs.plone.org/develop/plone/index.html)

* [**HTTP serving and traversing site data**](https://docs.plone.org/develop/plone/serving/index.html)

* HTTP request and response
* [**Previous**](https://docs.plone.org/develop/plone/serving/index.html)