# python-to-cl Documentation

*Release 1.0*

**Nikolai Matiushev**

**Mar 31, 2020**

# Popular Python libraries

Common Lisp is a nice and expressive language and this project is an effort to promote or improve CL libraries by comparing to the corresponding Python alternatives.

We attempt to document alternatives to few of the most popular Python libraries:

CHAPTER 1

# Dateutil

https://dateutil.readthedocs.io/en/stable/

The dateutil module provides powerful extensions to the standard datetime module, available in Python.

## 1.1 Features

- Computing of relative deltas (next month, next year, next monday, last week of month, etc);
- Computing of relative deltas between two given date and/or datetime objects;
- Computing of dates based on very flexible recurrence rules, using a superset of the iCalendar specification. Parsing of RFC strings is supported as well.
- Generic parsing of dates in almost any string format;
- Timezone (tzinfo) implementations for tzfile(5) format files (/etc/localtime, /usr/share/zoneinfo, etc), TZ environment string (in all known formats), iCalendar format files, given ranges (with help from relative deltas), local machine timezone, fixed offset timezone, UTC timezone, and Windows registry-based time zones.
- Internal up-to-date world timezone information based on Olson?s database.
- Computing of Easter Sunday dates for any given year, using Western, Orthodox or Julian algorithms.

## 1.2 CL alternatives

- https://github.com/dlowe-net/local-time
- https://github.com/enaeher/local-time-duration
- https://gist.github.com/perusio/6687883

## 1.3 Python to CL examples

https://dateutil.readthedocs.io/en/stable/examples.html

Similary to Python let's start with importing required libraries:

```
(ql:quickload :local-time)
(ql:quickload :local-time-duration)

(use-package :local-time)
```

Store some values:

```
(defparameter *now* (now))
(defparameter *today* (today))

(list *now* *today*)
```

Next month

```
(timestamp+ *now* 1 :month)
;; or
(adjust-timestamp *now* (offset :month 1))
```

```
@2020-04-24T10:48:48\.317722Z
```

Next month, plus one week

```
(adjust-timestamp *now* (offset :month 1) (offset :day 7))
```

```
@2020-04-30T10:48:48\.317722Z
```

Next month, plus one week, at 10am.

```
(adjust-timestamp *now* (offset :month 1) (offset :day 7) (set :hour 10))
```

```
@2020-04-30T10:48:48\.317722Z
```

Setting specific time fields similar to absolute relativedelta:

```
(adjust-timestamp *now* (set :year 1) (set :month 1))
```

```
@0001-01-24T10:48:48\.317722Z
```

Get the relative delta

```
(ltd:timestamp-difference (encode-timestamp 0 0 0 0 1 1 2018) *now*)
```

```
#<LOCAL-TIME-DURATION:DURATION [-813/-38928/-317722000] -116 weeks -1 days -10 hours -
↪48 minutes -48 seconds -317722000 nsecs>
```

One month before one year.

```
(adjust-timestamp *now* (offset :year 1) (offset :month -1))
```

```
@2021-02-24T10:48:48\.317722Z
```

How does it handle months with different numbers of days? Notice that adding one month will never cross the month boundary.

```
(adjust-timestamp (encode-timestamp 0 0 0 0 27 1 2003) (offset :month 1))
```

```
@2003-02-27T00:00:00\.000000Z
```

```
(adjust-timestamp (encode-timestamp 0 0 0 0 31 1 2003) (offset :month 1))
```

```
@2003-02-28T00:00:00\.000000Z
```

```
(adjust-timestamp (encode-timestamp 0 0 0 0 31 1 2003) (offset :month 2))
```

```
@2003-03-31T00:00:00\.000000Z
```

The logic for years is the same, even on leap years.

```
(adjust-timestamp (encode-timestamp 0 0 0 0 28 2 2000) (offset :year 1))
```

```
@2001-02-28T00:00:00\.000000Z
```

```
(adjust-timestamp (encode-timestamp 0 0 0 0 29 2 2000) (offset :year 1))
```

```
@2001-02-28T00:00:00\.000000Z
```

```
(adjust-timestamp (encode-timestamp 0 0 0 0 28 2 1999) (offset :year 1))
```

```
@2000-02-28T00:00:00\.000000Z
```

```
(adjust-timestamp (encode-timestamp 0 0 0 0 1 3 1999) (offset :year 1))
```

```
(adjust-timestamp (encode-timestamp 0 0 0 0 28 2 2001) (offset :year -1))
```

```
@2000-02-28T00:00:00\.000000Z
```

```
(adjust-timestamp (encode-timestamp 0 0 0 0 1 3 2001) (offset :year -1))
```

```
@2000-03-01T00:00:00\.000000Z
```

Next Friday

```
(adjust-timestamp *today* (offset :day-of-week :friday))
```

```
@2020-03-27T00:00:00\.000000Z
```

Last Friday of the month

```
(defun set-day-of-week (time day-of-week)
  "Adjust the timestamp to be the specifed day of the week, selects corresponding␣
↪preceeding date if timestamp's day of the week do not match the requirement."
  (let ((adjusted (adjust-timestamp time (offset :day-of-week day-of-week))))
    (if (timestamp>= time adjusted)
        adjusted
        (adjust-timestamp adjusted (offset :day -7)))))

(set-day-of-week (timestamp-maximize-part *today* :day) :friday)
```

```
@2020-03-27T23:59:59\.999999Z
```

Next Wednesday (it's today!)

```
(defun next-day-of-week (time day-of-week)
  "Adjust the timestamp to be the next specifed day of the week, selects␣
↪corresponding future date if timestamp's day of the week do not match the␣
↪requirement."
  (let ((adjusted (adjust-timestamp time (offset :day-of-week day-of-week))))
    (if (timestamp>= adjusted time)
        adjusted
        (adjust-timestamp adjusted (offset :day 7)))))

(let ((*today* (encode-timestamp 0 0 0 0 3 1 2018)))
  (next-day-of-week *today* :wednesday))
```

```
@2018-01-03T00:00:00\.000000Z
```

Next wednesday, but not today.

```
(let ((*today* (encode-timestamp 0 0 0 0 3 1 2018)))
  (next-day-of-week (adjust-timestamp *today* (offset :day 1)) :wednesday))
```

```
@2018-01-10T00:00:00\.000000Z
```

Following ISO year week number notation find the first day of the 15th week of 1997.

```
(set-day-of-week
 (adjust-timestamp
     (next-day-of-week
      (encode-timestamp 0 0 0 0 1 1 1997)
      :thursday)
   (offset :day (* 7 14)))
 :monday)
```

```
@1997-04-07T00:00:00\.000000Z
```

How long ago has the millennium changed?

```
(ltd:timestamp-difference *now* (encode-timestamp 0 0 0 0 1 1 2001))
```

```
#<LOCAL-TIME-DURATION:DURATION [7022/38928/317722000] 1003 weeks 1 day 10 hours 48␣
↪minutes 48 seconds 317722000 nsecs>
```

It works with dates too.

```
(ltd:timestamp-difference *today* (encode-timestamp 0 0 0 0 1 1 2001))
```

```
#<LOCAL-TIME-DURATION:DURATION [7022/0/0] 1003 weeks 1 day>
```

Obtain a date using the yearday:

```
(adjust-timestamp (timestamp-minimize-part *now* :day) (offset :day 260))
```

```
@2020-11-16T00:00:00\.000000Z
```

Leap year vs non-leap year:

```
(let ((leap (encode-timestamp 0 0 0 0 1 1 2000))
      (non-leap (encode-timestamp 0 0 0 0 1 1 2002)))

  (list (adjust-timestamp (timestamp-minimize-part leap :day) (offset :day 260))
        (adjust-timestamp (timestamp-minimize-part non-leap :day) (offset :day 260))))
```

CHAPTER 2

# Requests

https://requests.readthedocs.io/en/master/

Requests is the only Non-GMO HTTP library for Python, safe for human consumption.

## 2.1 Features

- Keep-Alive & Connection Pooling
- International Domains and URLs
- Sessions with Cookie Persistence
- Browser-style SSL Verification
- Automatic Content Decoding
- Basic/Digest Authentication
- Elegant Key/Value Cookies
- Automatic Decompression
- Unicode Response Bodies
- HTTP(S) Proxy Support
- Multipart File Uploads
- Streaming Downloads
- Connection Timeouts
- Chunked Requests
- .netrc Support

## 2.2 Common Lisp alternatives

- https://edicl.github.io/drakma/
- https://github.com/fukamachi/dexador

## 2.3 Python to CL examples

https://requests.readthedocs.io/en/master/user/quickstart/

Let's start with loading Drakma and Dexador libraries. Additionally we load a few useful libraries for our demo code.

```
(ql:quickload :quri)
(ql:quickload :drakma)
(ql:quickload :dexador)

(ql:quickload :jsown)
(ql:quickload :opticl)
(ql:quickload :flexi-streams)
```

### 2.3.1 Simple examples

Simple GET request, note that Dexador uses multiple return values to return request status, headers, etc.

```
(dexador:get "https://api.github.com/events")
```

```
[{"id":"11846698289","type":"PushEvent","actor":{"id":21087069,"login":"MozmarRobot",
↪"display_login":"MozmarRobot","g...
```

This is how you make an HTTP POST request:

```
(dexador:post "https://httpbin.org/post" :content '(("key" . "value")))
```

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "key": "value"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "9",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap",
    "X-Amzn-Trace-Id": "Root=1-5e79e4d6-3a32c1c071523ef0884867c8"
  },
  "json": null,
  "origin": "127.0.0.1",
  "url": "https://httpbin.org/post"
}
```

Other HTTP methods

```
(dexador:put "https://httpbin.org/put" :content '(("key" . "value")))
```


```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "key": "value"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "9",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap",
    "X-Amzn-Trace-Id": "Root=1-5e79e4e6-cdbfeadf5d99547ffe831aba"
  },
  "json": null,
  "origin": "127.0.0.1",
  "url": "https://httpbin.org/put"
}
```


```
(dexador:delete "https://httpbin.org/delete")
```


```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "0",
    "Host": "httpbin.org",
    "User-Agent": "Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap",
    "X-Amzn-Trace-Id": "Root=1-5e79e4f2-6c8844b8d737701857c59668"
  },
  "json": null,
  "origin": "127.0.0.1",
  "url": "https://httpbin.org/delete"
}
```


```
(multiple-value-bind (body status headers uri connection)
    (dexador:head "https://httpbin.org/get")
  (alexandria:hash-table-alist headers))
```


```
((access-control-allow-credentials . true) (access-control-allow-origin . *) (server .
↪ gunicorn/19.9.0) (connection . keep-alive) (content-length . 320) (content-type .␣
↪application/json) (date . Tue, 24 Mar 2020 10:46:26 GMT))
```


```
(multiple-value-bind (body status headers uri connection)
    (dexador:request "https://httpbin.org/get" :method :options)
  (alexandria:hash-table-alist headers))
```


```
((access-control-max-age . 3600) (access-control-allow-methods . GET, POST, PUT,␣
↪DELETE, PATCH, OPTIONS) (access-control-allow-credentials . true) (access-control-
↪allow-origin . *) (allow . GET, OPTIONS, HEAD) (server . gunicorn/19.9␣
↪(connection . keep-alive) (content-length . 0) (content-type . text/html;␣
↪charset=utf-8) (date . Tue, 24 Mar 2020 13:30:52 GMT))
```


(continues on next page)

(continued from previous page)

### 2.3.2 Passing parameters in URLs

If you wanted to pass key1=value1 and key2=value2 to httpbin.org/get, you would use the following code:

```
(let ((payload '(("key1" . "value1") ("key2" . "value2"))))
  (multiple-value-bind (body status headers uri connection)
      (dexador:get (quri:make-uri :defaults "https://httpbin.org/get" :query payload))
    uri))
```

```
#<QURI.URI.HTTP:URI-HTTPS https://httpbin.org/get?key1=value1&key2=value2>
```

You can also pass a list of items as a value:

```
(let ((payload '(("key1" . "value1") ("key2" . "value2") ("key2" . "value3"))))
  (multiple-value-bind (body status headers uri connection)
      (dexador:get (quri:make-uri :defaults "https://httpbin.org/get" :query payload))
    uri))
```

```
#<QURI.URI.HTTP:URI-HTTPS https://httpbin.org/get?key1=value1&key2=value2&key2=value3>
```

### 2.3.3 Response content

We can read the content of the server's response. Consider the GitHub timeline again:

```
(dexador:get "https://api.github.com/events")
```

```
[{"id":"11848108853","type":"PullRequestEvent","actor":{"id":9636382,"login":"rekols",
↪"display_login":"rekols","grava...
```

Dexador will automatically decode content from the server. Most unicode charsets are seamlessly decoded.

It is possible to get the guessed charset:

```
(multiple-value-bind (body status headers uri connection)
    (dexador:get "https://api.github.com/events")
  (dexador.encoding:detect-charset (gethash "content-type" headers) body))
```

```
:UTF-8
```

To manually fix encoding issues you can resort to geting raw binary data for further processing.

```
(dexador:get "https://api.github.com/events" :force-binary t)
```

```
(91 123 34 105 100 34 58 34 49 49 56 52 56 52 55 49 53 49 51 34 44 34 116 121 112 101␣
↪34 58 34 80 117 115 104 69 118 ...
```

### 2.3.4 Binary response content

You can also access the response body as bytes, for non-text requests:

```
(dexador:get "http://httpbin.org/image/jpeg")
```

```
(255 216 255 224 0 16 74 70 73 70 0 1 1 2 0 28 0 28 0 0 255 254 0 53 69 100 105 116␣
↪101 100 32 98 121 32 80 97 117 10...
```

The gzip and deflate transfer-encodings are automatically decoded for you.

For example, to create an image from binary data returned by a request, you can use the following code:

```
(ql:quickload 'opticl)


(opticl:read-image-stream
   (flexi-streams:make-in-memory-input-stream
     (dexador:get "http://httpbin.org/image/jpeg"))
   "jpeg")
```

```
#3A(((3 0 0)
     (4 3 1)
     (0 1 0)
     (0 2 0)
     (1 1 0)
     (2 2 0)
     (0 2 0)
     (0 3 0)
     (0 0 0)
...
```

### 2.3.5 JSON response contents

Dexador doesn't provide built-in support for decoding JSON. Please use other libraries to handle parsing i.e. https://github.com/madnificent/jsown

```
(jsown:parse
  (dexador:get "https://api.github.com/events"))
```

```
((:OBJ (id . 11849548801) (type . IssueCommentEvent) (actor :OBJ (id . 8228920)␣
↪(login . JakeRL) (display_login . Jak...
```

### 2.3.6 Raw response content

Dexador doesn't provide access to raw socket streams. But you can get binary stream for decompressed body data.

```
(dexador:get "https://api.github.com/events" :force-binary t :want-stream t)
```

```
#<DEXADOR.KEEP-ALIVE-STREAM:KEEP-ALIVE-STREAM {10032B80E3}>
200
#<HASH-TABLE :TEST EQUAL :COUNT 24 {10032B5343}>
#<QU...
```

### 2.3.7 Custom headers

If you'd like to add HTTP headers to a request, simply pass in an alist to the `headers` parameter.

For example, let's specify user-agent:

```
(dexador:get "http://httpbin.org/headers" :headers '(("user-agent" . "my-app/0.0.1")␣
↪(:foo . :bar)))
```

```
{
  "headers": {
    "Accept": "*/*",
    "Content-Length": "0",
    "Foo": "BAR",
    "Host": "httpbin.org",
    "User-Agent": "my-app/0.0.1",
    "X-Amzn-Trace-Id": "Root=1-5e7a2861-7310e3606d01dbac675dd3dc"
  }
}
```

Note how Dexador automatically converts header names to capitalised kebab case.

### 2.3.8 More complicated POST requests

Typically, you want to send some form-encoded data — much like an HTML form. To do this, simply pass an alist to the `content` argument. Your alist of data will automatically be form-encoded when the request is made:

```
(dexador:post "http://httpbin.org/post" :content '(("key1" . "value1") ("key2" .
↪"value2")))
```

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "key1": "value1",
    "key2": "value2"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "23",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap",
    "X-Amzn-Trace-Id": "Root=1-5e7a2e4c-cbcbf430b6beb930e5d8f450"
  },
  "json": null,
  "origin": "127.0.0.1",
  "url": "http://httpbin.org/post"
}
```

The `content` argument can also have multiple values for each key. This is particularly useful when the form has multiple elements that use the same key:

```
(dexador:post "http://httpbin.org/post" :content '(("key1" . "value1") ("key1" .
↪"value2") ("key2" . "value3")))
```

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "key1": [
      "value1",
      "value2"
    ],
    "key2": "value3"
  },
  "headers": {
    "Accept": "*/*",
    "Content-Length": "35",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap",
    "X-Amzn-Trace-Id": "Root=1-5e7a2f3d-9a58a53d4103ce8508cec6cc"
  },
  "json": null,
  "origin": "127.0.0.1",
  "url": "http://httpbin.org/post"
}
```


There are times that you may want to send data that is not form-encoded. If you pass in a string instead of an alist, that data will be posted directly.

```
(dexador:post "http://httpbin.org/post"
  :content (jsown:to-json '(:OBJ ("key" . "value")))
  :headers '((:content-type . "application/json")))
```


```
{
  "args": {},
  "data": "{\"key\":\"value\"}",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "15",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap",
    "X-Amzn-Trace-Id": "Root=1-5e7a3175-9b90305f1ecde7d26a8c4517"
  },
  "json": {
    "key": "value"
  },
  "origin": "127.0.0.1",
  "url": "http://httpbin.org/post"
}
```


### 2.3.9 POST a Multipart-Encoded File

Dexador directly supports sending Multipart-encoded files.

```
(dexador:post "http://httpbin.org/post"
  :content '(("hello.txt" . #p"hello.txt")))
```

```
{
  "args": {},
  "data": "",
  "files": {
    "hello.txt": "Hello world!\n"
  },
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "149",
    "Content-Type": "multipart/form-data; boundary=QksivVtcwqyA",
    "Host": "httpbin.org",
    "User-Agent": "Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap",
    "X-Amzn-Trace-Id": "Root=1-5e7a325a-ade74fbd4dbf683558c0e642"
  },
  "json": null,
  "origin": "127.0.0.1",
  "url": "http://httpbin.org/post"
}
```

### 2.3.10 Response Status codes

Status code is returned as one of the multiple values from Dexador request call:

```
(multiple-value-bind (body status headers url connection) (dexador:get "http://
↪httpbin.org/get")
  status)
```

```
200
```

Bad requests will signal a `http-request-failed` condition

```
(handler-case (dex:get "https://httpbin.org/status/404")
  (dex:http-request-failed (e)
    (format nil  "The server returned ~D" (dex:response-status e))))
```

```
The server returned 404
```

You can handle more specialized conditions

```
(handler-case (dex:get "https://httpbin.org/status/400")
  (dex:http-request-bad-request (e)
    (format nil  "Bad reqest was sent to server: ~D" (dex:response-status e)))
  (dex:http-request-failed (e)
    (format nil  "The server returned ~D" (dex:response-status e))))
```

```
Bad reqest was sent to server: 400
```

```
(handler-case (dex:get "https://httpbin.org/status/404")
  (dex:http-request-not-found (e)
```

(continues on next page)

(continued from previous page)

```
    (format nil  "Page not found: ~D" (dex:response-status e)))
  (dex:http-request-failed (e)
    (format nil  "The server returned ~D" (dex:response-status e))))
```

```
Page not found: 404
```

You can ignore specific conditions

```
(handler-bind ((dexador:http-request-not-found #'dexador:ignore-and-continue))
  (dexador:get "https://httpbin.org/status/404"))
```

Or retry the request.

```
(let ((retry-request (dex:retry-request 5 :interval 3)))
  (handler-bind ((dex:http-request-failed retry-request))
    (dex:get "https://httpbin.org/status/404")))
```

This will result in condition afer about 15 seconds.

```
An HTTP request to "https://httpbin.org/status/404" returned 404 not found.
   [Condition of type DEXADOR.ERROR:HTTP-REQUEST-NOT-FOUND]

Restarts:
 0: [RETRY-REQUEST] Retry the same request.
 1: [IGNORE-AND-CONTINUE] Ignore the error and continue.
 2: [RETRY] Retry SLIME evaluation request.
 3: [*ABORT] Return to SLIME's top level.
 4: [ABORT] abort thread (#<THREAD "worker" RUNNING {10017C1793}>)

Backtrace:
  0: (DEXADOR.ERROR:HTTP-REQUEST-FAILED 404 :BODY "" :HEADERS #<HASH-TABLE :TEST␣
↪EQUAL :COUNT 7 {1001AF01D3}> :URI #<QURI.URI.HTTP:URI-HTTPS https://httpbin.org/
↪status/404> :METHOD :GET)
  1: (DEXADOR.BACKEND.USOCKET:REQUEST #<unavailable argument> :METHOD :GET)
  2: ((LAMBDA ()))
```

### 2.3.11 Response headers

We can view the server's response headers:

```
(multiple-value-bind (body status headers uri connection)
    (dexador:head "https://httpbin.org/get")
  (alexandria:hash-table-alist headers))
```

```
((access-control-allow-credentials . true) (access-control-allow-origin . *) (server .
↪ gunicorn/19.9.0) (connection . keep-alive) (content-length . 320) (content-type .␣
↪application/json) (date . Tue, 24 Mar 2020 17:10:43 GMT))
```

Since header names are case insensitive keys in the headers hash table are converted to lower case.

### 2.3.12 Cookies

Dexador adopts https://github.com/fukamachi/cl-cookie for its cookie management. All functions takes a `cookie-jar` instance at `:cookie-jar`.

```
(defvar *cookie-jar* (cl-cookie:make-cookie-jar))

;; setting cookies
(dex:head "https://mixi.jp" :cookie-jar *cookie-jar*)
```

```
;; getting cookies
(dex:head "https://mixi.jp" :cookie-jar *cookie-jar*)
*cookie-jar*
```

```
#S(CL-COOKIE:COOKIE-JAR
   :COOKIES (#S(CL-COOKIE:COOKIE
                :NAME "_auid"
                :VALUE "4265774dfa8b2c3d23a821304b8fe9f6"
                :EXPIRES 3857131561
                :PATH NIL
                :DOMAIN ".mixi.jp"
                :SECURE-P NIL
                :HTTPONLY-P NIL
                :ORIGIN-HOST "mixi.jp")
             #S(CL-COOKIE:COOKIE
                :NAME "_auid_xsite"
                :VALUE "4265774dfa8b2c3d23a821304b8fe9f6"
                :EXPIRES 3857131561
                :PATH NIL
                :DOMAIN ".mixi.jp"
                :SECURE-P T
                :HTTPONLY-P T
                :ORIGIN-HOST "mixi.jp")
             #S(CL-COOKIE:COOKIE
                :NAME "_lcp"
                :VALUE "5787e0cbb4d7746f961ed16940837ac5"
                :EXPIRES 3794146153
                :PATH NIL
                :DOMAIN ".mixi.jp"
                :SECURE-P NIL
                :HTTPONLY-P NIL
                :ORIGIN-HOST "mixi.jp")))
```

### 2.3.13 Redirection and History

Dexador automatically follows redirects on GET and HEAD requests. You can limit the count of redirection by specifying `:max-redirects` with an integer. The default value is 5.

```
(multiple-value-bind (body status headers uri connection)
    (dex:get "http://httpbin.org/redirect/2")
  (list status uri body))
```

```
(200 #<QURI.URI.HTTP:URI-HTTP http://httpbin.org/get> "{
  \"args\": {},
  \"headers\": {
    \"Accept\": \"*/*\",
    \"Content-Length\": \"0\",
    \"Host\": \"httpbin.org\",
    \"User-Agent\": \"Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap\",
```

(continues on next page)

(continued from previous page)

```
    \"X-Amzn-Trace-Id\": \"Root=1-5e7a456e-7fd198882e529df8fad9af50\"
  },
  \"origin\": \"127.0.0.1\",
  \"url\": \"http://httpbin.org/get\"
}
")
```

```
(multiple-value-bind (body status headers uri connection)
    (dex:get "http://httpbin.org/redirect/3" :max-redirects 2)
  (list status uri body))
```

```
(302 #<QURI.URI.HTTP:URI-HTTP http://httpbin.org/relative-redirect/1> "")
```

You can use forth returned parameter to get the URL of the final redirect location.

Dexador doesn't track the history of responses.

### 2.3.14 Timeouts

You can tell Dexador to stop waiting for a connection after `connect-timout` and waiting to read a response after `read-timeout` number of seconds.

```
(dex:get "http://httpbin.org/delay/5")
```

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "0",
    "Host": "httpbin.org",
    "User-Agent": "Dexador/0.9.14 (SBCL 2.0.2); Linux; 4.14.24-qnap",
    "X-Amzn-Trace-Id": "Root=1-5e7a46ad-e273ae4e4c482efef2354f24"
  },
  "origin": "127.0.0.1",
  "url": "http://httpbin.org/delay/5"
}
```

```
(handler-case (dex:get "http://httpbin.org/delay/5" :read-timeout 3)
  (error (c)
    c))
```

```
#<SB-SYS:IO-TIMEOUT {100E06A383}>
```