

# The HTTP Plug-in

## Table of contents

1 What's it for?.....	2
2 Controlling the HTTPPlugin.....	2
2.1 Levels of Control.....	2
2.2 Importing the HTTPPluginControl.....	3
2.3 Setting HTTPClient Authorization Module.....	3
2.4 Setting an HTTP proxy.....	3
2.5 Setting HTTP Headers.....	4
2.6 Setting Encoding.....	4
2.7 Setting Redirect Behaviour.....	4
2.8 Setting Local Address.....	5
2.9 Setting Timeout Value.....	5
2.10 Setting Cookie Behaviour.....	5
2.11 Automatic decompression of gzipped responses.....	5
2.12 Streaming requests and response.....	6
3 Using HTTPUtilities.....	6
3.1 Setting Basic Authorization.....	6
3.2 Getting the Last Response.....	6
3.3 Getting a Token Value from a Location URI.....	6
3.4 Getting a Token Value from a URI in the Body of the Response .....	7

## 1 What's it for?

The HTTPPlugin is a mature plug-in for testing HTTP services. It has a number of utilities useful for HTTP scripts as well as a tool, the [TCPProxy](#) ( `../g3/tcpproxy.html#HTTPPluginTCPProxyFilter` ), which allows HTTP scripts to be automatically recorded. Recorded scripts are often customised, for example to simulate multiple users. This requires you to know a little about writing [scripts](#) ( `../g3/scripts.html` ) .

The HTTPPlugin is built into The Grinder and is automatically initialised whenever a script imports one of its classes. For example:

```
from net.grinder.plugin.http import HTTPRequest
```

The key class provided by the plug-in is [HTTPRequest](#) ( `../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html` ) . The best way to see how to use this class is to record a script with the TCPProxy.

The plug-in wires itself into The Grinder script life cycle. It maintains a cache of connections and cookies for each worker thread which it resets at the beginning of each run. Each run performed by a worker thread simulates a browser session carried out by a user. Resetting a thread's cookies at the beginning of a run will cause server applications that use cookie-based tracking to create a new session.

If your server application uses some other mechanism for session tracking (e.g. URL rewriting or hidden parameters), the script will have to capture and resend the appropriate token. The TCPProxy goes to some lengths to identify and record these tokens.

If an `HTTPRequest` is instrumented with a `Test`, the plug-in contributes additional statistics, including the HTTP status code, the response body length, and additional connection timing information. These statistics appear in the console and are recorded to the process data log. If several `HTTPRequest`s are instrumented within the same `Test` (e.g. they are called within an instrumented function), the status code of the last response is recorded.

## 2 Controlling the HTTPPlugin

The behaviour of the plug-in can be controlled from within scripts run by The Grinder through the use of the [HTTPPluginControl](#) ( `../g3/script-javadoc/net/grinder/plugin/http/HTTPPluginControl.html` ) facade.

### 2.1 Levels of Control

There are three levels of control of the behaviour of the HTTPPlugin that the HTTPPluginControl facade gives you access to:

1. **Default Connection Behaviour**
  - Method: `getConnectionDefaults`
  - Returns a `HTTPPluginConnection` that can be used to set the default behaviour of new connections.
2. **Thread Connection Behaviour**
  - Method: `getThreadConnection`
  - Returns a `HTTPPluginConnection` for a particular URL.

- The resulting `HTTPPluginConnection` is valid for the current thread and the current run. It can be used to set specific authentication details, default headers, cookies, proxy servers, and so on for the current thread/run on a per-URL basis.
  - This method will throw a `GrinderException` if not called from a worker thread.
3. **Thread `HTTPClient` Context Object Behaviour**
- Method: `getThreadHTTPClientContext`
  - Returns the `HTTPClient` context object for the calling worker thread. This is useful when calling `HTTPClient` methods directly, e.g. `CookieModule.listAllCookies(Object)`.
  - This method will throw a `GrinderException` if not called from a worker thread.

## 2.2 Importing the `HTTPPluginControl`

Place the following line at the top of your grinder script along with your other import statements

```
from net.grinder.plugin.http import HTTPPluginControl
```

## 2.3 Setting `HTTPClient` Authorization Module

The `HTTPClient` Authorization module is no longer enabled by default because it prevents raw authentication headers being sent through. The module also slows things down as `HTTPClient` must parse responses for challenges.

Users who still wish to use the `HTTPClient` Authorization module can enable it using:

```
control = HTTPPluginControl.getConnectionDefaults()
control.setUseAuthorizationModule(1)
```

The authentication details can be set using the [AuthorizationInfo](#) (`../g3/script-javadoc/HTTPClient/AuthorizationInfo.html`) API. `HTTPClient` maintains authentication information separately in each context, so the API must be called by each worker thread. See the [Digest Authentication sample](#) (`../g3/script-gallery.html#digestauthentication.py`) in the script gallery, as well as the example in the next section.

## 2.4 Setting an HTTP proxy

Should you need to specify an HTTP proxy to route requests through the following code can be used to specify the default proxy.

```
control = HTTPPluginControl.getConnectionDefaults()
control.setProxyServer("localhost", 8001)
```

HTTP proxies can also be specified at the thread connection level. This is useful to set proxies on a per URL basis.

```
proxyURL1 = HTTPPluginControl.getThreadConnection("http://url1")
proxyURL2 = HTTPPluginControl.getThreadConnection("http://url2")
proxyURL1.setProxyServer("localhost", 8001)
proxyURL2.setProxyServer("localhost", 8002)
```

If the HTTP proxy requires authentication, enable the HTTPClient Authorization Module, as described in the previous section. Having so, each worker thread can set up the appropriate authentication details using the [AuthorizationInfo](#) ( `../g3/script-javadoc/HTTPClient/AuthirizationInfo.html`) API. For example:

```
from net.grinder.plugin.http import HTTPRequest, HTTPPluginControl
from HTTPClient import AuthorizationInfo

defaults = HTTPPluginControl.getConnectionDefaults()
defaults.useAuthorizationModule = 1
defaults.setProxyServer("localhost", 3128)

class TestRunner:
    def __init__(self):
        AuthorizationInfo.addBasicAuthorization("localhost",
                                              8001,
                                              "My Proxy Realm",
                                              "joeuser",
                                              "pazzword",

        HTTPPluginControl.getThreadHTTPClientContext()

    def __call__(self):
        # ...
```

## 2.5 Setting HTTP Headers

---

The HTTPPlugin allows you to set the HTTP Headers sent with requests. The method takes the settings as header-name/value pairs

```
control = HTTPPluginControl.getConnectionDefaults()
control.setDefaultHeaders(NVPair("header-name", "value"),)
```

Typical headers you might want to set here are Accept and its Accept-\* relatives, Connection, From, User-Agent, etc.

For example to disable persistent connections:

```
control = HTTPPluginControl.getConnectionDefaults()
control.setDefaultHeaders(NVPair("Connection", "close"),)
```

## 2.6 Setting Encoding

---

Encoding for Content or for Transfer can be switched on and off using boolean flags

```
control = HTTPPluginControl.getConnectionDefaults()
control.setUseContentEncoding(0)
control.setUseTransferEncoding(1)
```

## 2.7 Setting Redirect Behaviour

---

Setting the HTTPPlugin behaviour with regards to following redirects can be switched on and off using boolean flags

```
control = HTTPPluginControl.getConnectionDefaults()
control.setFollowRedirects(0)
```

## 2.8 Setting Local Address

---

Should you be conducting your tests on a server with multiple network interfaces you can set the local IP address used by the HTTPPlugin for outbound connections.

```
control = HTTPPluginControl.getConnectionDefaults()
control.setLocalAddress("192.168.1.77")
```

## 2.9 Setting Timeout Value

---

The timeout value for used for creating connections and reading responses can be controlled via the HTTPPlugin. The time is specified in milliseconds.

The following example sets a default timeout value of 30 seconds for all connections.

```
control = HTTPPluginControl.getConnectionDefaults()
control.setTimeout(30000)
```

## 2.10 Setting Cookie Behaviour

---

Setting the HTTPPlugin behaviour with regards to whether cookies are used or not can be switched on and off using boolean flags

```
control = HTTPPluginControl.getConnectionDefaults()
control.setUseCookies(0)
```

## 2.11 Automatic decompression of gzipped responses

---

For load testing, its often not practical to uncompress the response. It's simply too expensive in CPU terms to do all that decompression in the client worker process. This doesn't mean you can't test a server that compresses its responses, just that you can't parse the responses in the script.

On the other hand, there are times you may want to do this. The Grinder supports decompression which it inherits from the HTTPClient library, you just need to enable it. If your server encrypts the content and sets a Content-Encoding header that starts with one of { gzip, deflate, compress, identity }, you can automatically decrypt the responses by adding the following lines to the beginning of your script:

```
from net.grinder.plugin.http import HTTPPluginControl
connectionDefaults = HTTPPluginControl.getConnectionDefaults()
connectionDefaults.useContentEncoding = 1
```

Similarly, if your server sets a Transfer-Encoding header that starts with one of { gzip, deflate, compress, chunked, identity }, you can enable the HTTPClient Transfer Encoding Module with `connectionDefaults.useTransferEncoding = 1`.

There is no support for automatically decrypting things based on their Content-Type (as opposed to Content-Encoding, Transfer-Encoding). Your browser doesn't do this, so neither should The Grinder. If you really want to do this, you can use Java or Jython decompression libraries from your script.

## 2.12 Streaming requests and response

The [HTTPRequest](#) ( [../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html](#)) class has support for sending request data from a stream. This allows an arbitrarily large amount of data to be sent without requiring a corresponding amount of memory. To do this, use these versions of the [POST](#) ( [../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html#POST\(java.lang.String, java.io.InputStream\)](#)), [PUT](#) ( [../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html#POST\(java.lang.String, java.io.InputStream\)](#)), [OPTIONS](#) ( [../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html#POST\(java.lang.String, java.io.InputStream\)](#)), methods.

`HTTPRequest` allows the response body to be handled as a stream. Refer to the Javadoc for the [setReadResponseBody](#) ( [../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html#setReadResponseBody\(boolean\)](#)) method for more details.

## 3 Using HTTPUtilities

The `HTTPPlugin` provides an `HTTPUtilities` class:

```
net.grinder.plugin.http.HTTPUtilities
```

This class has several methods which are useful for HTTP scripts.

### 3.1 Setting Basic Authorization

The `HTTPUtilities` class can create an `NVPair` for an HTTP Basic Authorization header using the following method:

```
httpUtilities = HTTPPluginControl.getHTTPUtilities()
httpUtilities.basicAuthorizationHeader('username', 'password')
```

Include the header with each `HTTPRequest` that requires the authentication.

```
request101.GET('/', (),
    ( httpUtilities.basicAuthorizationHeader('prelive', 'g3tout'), ))
```

### 3.2 Getting the Last Response

The `HTTPUtilities` class can return the response for the last request made by the calling worker thread using the following method:

```
httpUtilities = HTTPPluginControl.getHTTPUtilities()
httpUtilities.getLastResponse()
```

This returns the response, or `null` if the calling thread has not made any requests.

This must be called from a worker thread, if not it throws a `GrinderException`.

### 3.3 Getting a Token Value from a Location URI

The `HTTPUtilities` class can return the value for a path parameter or query string name-value token with the given `tokenName` in a Location header from the last response. If there are multiple matches, the first value is returned. This utility can be invoked using the following method:

```
httpUtilities = HTTPPluginControl.getHTTPUtilities()  
httpUtilities.valueFromLocationURI(tokenName)
```

If there is no match, an empty string is returned rather than `null`. This makes scripts more robust (as they don't need to check the value before using it), but they lose the ability to distinguish between a missing token and an empty value.

This must be called from a worker thread, if not it throws a `GrinderException`.

### 3.4 Getting a Token Value from a URI in the Body of the Response

---

The `HTTPUtilities` class can return the value for a path parameter or query string name-value token with the given `tokenName` in a URI in the body of the last response. If there are multiple matches, the first value is returned. This utility can be invoked using the following method:

```
httpUtilities = HTTPPluginControl.getHTTPUtilities()  
httpUtilities.valueFromBodyURI(tokenName)
```

This returns the first value if one is found, or `null`.

This must be called from a worker thread, if not it throws a `GrinderException`.