

# HTTP Server Classes for REALbasic 2006r3

<http://www.thezaz.com/opensource/realbasic/server/http/>  
by The ZAZ Studios

## Introduction

HTTP is a simple protocol to implement, and is used every day by nearly every computer user. It's a method for providing data on-demand. Although used less frequently, it can be used to manage a file system through the more uncommon methods, such as PUT and DELETE.

These classes take another go at implementing an HTTP server in REALbasic. Our goal was to make it easy to integrate into existing projects, as well as make it easy to serve both static and dynamic pages. It was important to support powerful features, such as sessions and cookies, while not requiring developers to write rbscript files to process dynamic pages.

## Usage

The easiest way to learn the ropes is to launch the demonstration project. The code you are interested in is short and simple, which is precisely what you're going to implement in your own project.

You'll need to create the server first. Nothing complicated, just call "new httpserversocket" and you're done. It would be a good idea, of course, to assign a port at this point as well. You can either specify the port as a parameter during creation, or simply setting the "port" variable later. UNIX-based operating systems (Linux & Mac OS X) require admin access to bind to ports below 1024, so pick a port higher unless you're certain your application will be run under correct permissions.

Before you start serving pages, you'll want to add some urls to the server. HTTPServerSocket has an AddURL method, which takes a url string, and either a FolderItem or an Object. For example:

```
MyHTTPServerSocket.AddURL("/", volume(0))
MyHTTPServerSocket.AddURL("/form", MySimpleForm)
```

Don't add your entire boot drive, by the way. This would expose many problems and take much time to process.

You may also specify how the server searches for "index" files. By default, a url ending in "/" will search for the url same url appended with "index.html". You can customize this behavior by modifying the HTTPServerSocket.IndexFiles array.

Once setup, simply call HTTPServerSocket.Listen, and your server is running.

There is also an included Class Interface, HTTPRequestHandler. Any object you wish to serve must implement this interface. It only requires a single method, "handleRequest" which gets passed an HTTPRequestContext. In this event, the class must modify the attributes of the HTTPRequestContext. This is where the true power comes in.

If you leave the context alone, it will return a 404 (File Not Found) error. Usually, you don't want that. The context comes pre-loaded with all the data you'll need to process your page: Cookies, Request Headers, Variables, Sessions, and Response Headers. For example, to read the value store in a cookie "mycookie", simply call context.cookie("mycookie"). If there is a cookie, you'll get the data. If not, you get an empty string. No exceptions to worry about! The same goes for Header and Variable. You can obtain a session by calling context.session, which provides an HTTPSession object. If a new session needs to be created, one will be done for you and the cookie sent automatically. The same session will automatically be reloaded during later requests.

You will also need to set the context's status code. Normally, this is 200 (OK). This is easy using the HTTPServer.kStatus... constants. And should you need it, there is an entity variable. This contains raw data sent from a PUT or POST method.

## Class Documentation

Ok, assuming you want things more technical, we'll give that as well. Only information you'll actually use is here.

### Classes

#### HTTPServerSocket

##### Methods

##### AddURL (URL As String, File As FolderItem)

Used for linking a file to an absolute url, such as "/" to volume(0)

##### AddURL (URL As String, Handler As HTTPRequestHandler)

Used for linking any object which implements HTTPRequestHandler to an absolute url.

##### RemoveURL (URL As String)

Remove a previously linked url.

##### Socket (Index As Integer) As HTTPClientHandler

Retrieve an HTTPClientHandler socket by index. HTTPClientHandler is undocumented and it is not recommended to tamper with it.

##### SocketCount () As Integer

Number of HTTPClientHandler sockets in the internal socket pool. Does not represent active or connected sockets, simply sockets that are ready to use, are in use, or have been used.

##### Properties

##### IndexFiles() As String

Ordered array containing the index file names to be searched for when omitted in directory urls. For example, the url "/" would only look for "/index.html" if the default array contained only "index.html" – which it does.

#### HTTPSession

##### Properties

##### Value (Key As String) As Variant

Place an object into, or read out of the session using this pseudo-property. Session is stored locally, so it is safe to add objects and other data to the session.

#### HTTPRequestContext

##### Methods

##### Cookie (Name As String) As String

Retrieve the value of the cookie specified. Returns "" if no cookie exists. To create a cookie, use SetCookie.

##### Header (Name As String) As String

Retrieve an HTTP request header by name. The HTTPServer module has kHeader... constants which will make this easier and avoid typos. To set a response header, use the Headers dictionary.

##### Print (Text As String)

Append Text to the response buffer. This is a byte-for-byte append, no line endings or other conversions are performed.

##### Session () As HTTPSession

Returns a session for use. If a session did not exist for this request, a new one is created and header automatically sent. Otherwise, returns the session created in a previous request. Will always return an HTTPSession object.

##### SetCookie (Name As String, [Value As String], [Expire As Date], [Path As String], [Domain As String], [Secure As Boolean])

Adds a Set-Cookie (HTTPServer.kHeaderSetCookie) header to your response which will add a cookie to the client. To remove a cookie, send the exact same parameters as the original, however set the expiration date to the past.

##### Variable (Name As String) As String

Return the variable specified. Automatically joins values and returns both GET and POST variables for you. If there is no variable by the requested name, "" is returned instead.

##### Properties

##### Buffer As String

The raw code that will be sent to the client. Using this property, the output can be modified, searched, or even replaced. Typically, however, you should use

Print() for creating output.

#### Entity As String

In the case of POST and more uncommon HTTP methods, an entity is sent containing binary data. Should you need to get at this data, here it is.

#### Headers As Dictionary

The headers which will be sent back to the client. Your response will work perfectly fine if you do not modify the headers, but some great features can be unlocked by using them.

#### Status Code As Integer

In most requests, you'll set this to 200 (OK – `HTTPServer.kStatusOK`) – but there are tons of response codes, and we've added all of them to the `HTTPServer` module. If you do not change this value in your request, the client will receive a 404 (File Not Found – `HTTPServer.kStatusFileNotFound`) and you likely don't want that.

### Modules

#### HTTPServer

##### Methods

#### URLDecode (URL As String) As String

Convert a string from RFC 1738 compliance into normal language.

#### URLEncode (URL As String) As String

Convert a string into RFC 1738 compliance. This is required in URLs and HTTP variables. The `HTTPRequestContext` automatically decodes the values for you, and the `HTTPClientHandler` socket encodes them in the response.

### Class Interfaces

#### HTTPRequestHandler

Any class which plans to respond to an HTTP request, must implement this interface. Luckily, it's simple.

##### Methods

#### HandleRequest (Context As HTTPRequestContext)

Your class will be provided with an `HTTPRequestContext`. Modify it, and when the method is complete, the response will be sent to the client. Output is not started until the entire method is complete, so headers may be sent to the client at any time.

### Contact Information

You can reach me via e-mail at [thom@thezaz.com](mailto:thom@thezaz.com).

### License

You may use these classes as you need, they're open source. You may only redistribute if:

- 1) The classes and documentation are unmodified from their original form. And
- 2) The documentation is included.

If you make changes, e-mail them to me. If they work out well, I'll roll them into a new release, and give credit where credit is due. But please, don't start distributing the modified versions - that's just a mess.