RexxHttp
Servlet programming in Rexx

José María Blasco

Version 0.1 20061101

RexxHttp: Servlet programming in Rexx
Version 0.1 20061101

Contents

Chapter 1

Introduction

RexxHttp is a ooRexx (and Object Rexx) Servlet and Rexx Server Pages (RSP) processor ("the servlet processor" or "the servlet engine") that runs under CGI and Mod_Rexx under the Apache HTTP server and can be made to run under other servers using CGI (for example, Microsoft IIS). RexxHttp runs under Windows and OS/2, and should run under any version of Unix for which there is a corresponding implementation of ooRexx or Object Rexx and Apache.

RexxHttp can be used to develop CGI programs and Mod_Rexx programs, and provides a convenient, uniform, object-oriented abstraction of the HTTP request/response model. RexxHttp can also be used to develop *portable servlets*, that is, servlets which run unmodified under several or all platforms supporting RexxHttp. This means that a servlet written following the portability guidelines[1] can be developed, for example, using OS/2 Object Rexx under Apache 1.3.35/Mod_Rexx 1.2.0, and deployed as a CGI application under ooRexx and Microsoft IIS 5.5 for Windows XP Professional, unchanged.[2]

RexxHttp automatically creates a *request* object which encapsulates the details of the HTTP request, a *response* object to receive the HTTP response headers, and an associated *output stream* where the contents of the HTTP response will be written. This output stream is a subclass of the builtin Stream class and implements a buffering mechanism, thus allowing for the modification of the response headers, as long as the response is not committed and the output stream has not been flushed.

RexxHttp also defines a *RSP compiler interface* (see *Page compiler interface* on page 51). Pages can be written which contain a mixture of HTML and Rexx code, have non-HTML tags translated to Rexx function calls (like JSPs), or whatever a compiler designer decides to implement. Current page compilers (i.e., Mod_Rexx's RSPCOMP and REXXTAGS) run with little modifications under RexxHttp.

RexxHttp is Open Source, and is distributed under the Common Public License contained in Appendix E, *Common Public License Version 1.0*, on page 63.

[1] See *Writing portable servlets* on page 47.

[2] IIS support is experimental in this release.

1.1 Structure of this manual

This manual is structured as follows: Chapter 1 is this Introduction.

Chapter 2, *Installation guide*, on page 3 contains detailed installation instructions for HttpRexx under Apache/CGI and Apache/Mod_Rexx, and defines the way to pass variables to HttpRexx.

Chapter 3, *Rationale*, on page 11 contains a historical note and detailed exposition of the design goals, decisions and compromises that had to be made to be able to write RexxHttp.[3]

Chapter 4, *Tutorial*, on page 19 contains a short RexxHttp tutorial. You can use it as an introduction, to get a feeling of what can be done with RexxHttp.

Chapter 5, *Class Reference*, on page 23 is the reference manual for RexxHttp. It contains complete descriptions of all the classes introduced by RexxHttp.

Chapter 6, *Writing portable servlets*, on page 47 details the guidelines to follow if you want to write portable servlets.

Chapter 7, *Page compiler interface*, on page 51 describes a simple interface to be used by Rexx Server Pages compiler writers so that their compilers can be used under RexxHttp.

The Appendices include OS/2-specific infomation, preliminary information about running RexxHttp under Microsoft IIS, information about running Rexx Server Pages compilers under RexxHttp, and the license information.

1.2 Contact information

Email me at `jose.maria.blasco@epbcn.com`.

[3]While on the one hand I understand that I take the risk of exposing my ignorance by writing such a chapter, on the other hand I think that hiding the productive process under the shine of a finished product is bad practice: it discourages discussion, makes the reasons behind the design of the product somewhat incomprehensible, and deters collaboration — and all of those are bad things for an Open Source product.

Chapter 2

Installation guide

2.1 Package contents

Here is a short description of the package contents:

`RexxHttp.rex` in the directory `/main` is the main RexxHttp processing module.

`Http.Cookie.cls` in the directory `/support` implements the Http.Cookie class.

`Http.OutputStream.cls` in the directory `/support` implements the Http.OutputStream class.

`Http.Request.cls` in the directory `/support` implements the Http.Request class.

`Http.Response.cls` in the directory `/support` implements the Http.Response class.

All the `.cls` files containes in the `support` are referred collectively as "the support classes'.
`Http.Aux.SimpleMutableBuffer.cls` in the directory `/support` contains a partial implementation of the MutableBuffer class for OS/2 and backlevel versions of Object Rexx.

`rhrspcmp.rex` and `rhrxtags.rex` in the directory `/support` are RexxHttp-ready versions of the RSPCOMP and REXXTAGS Rexx Server Pages compilers. See *Running RSPCOMP under RexxHttp* on page 59 and *Running REXXTAGS under RexxHttp* on page 61 for details.

`test.html` in the `/test` directory is the sample RexxHttp test page. You can use it to test RexxHttp features.

Other directories contain sample programs, including all the sample programs of the Tutorial.

2.2 Basic installation

2.2.1 Install the support classes

Put

```
Http.Cookie.cls,
Http.OutputStream.cls,
Http.Request.cls,
Http.Response.cls and
Http.Aux.SimpleMutableBuffer.cls
```

somewhere in your path so that RexxHttp.rex can find the files.

2.2.2 Install RexxHttp.rex

2.2.2.1 Under CGI

Install `RexxHttp.rex` in a CGI-enabled directory.

2.2.2.2 Under Mod_Rexx

Install RexxHttp.rex in a directory enabled to execute `.rex` files as Mod_Rexx programs (i.e., one to which the

```
AddType application/x-httpd-rexx-script .rex
```

Mod_Rexx directive applies; see your Mod_Rexx documentation for details).

Note: It might happen that you want to use RexxHttp *both* as a CGI and as a Mod_Rexx application *in the same server*. For example, you might want to process `.html` files as CGI files and `.htm` files as Mod_Rexx files. One possible setup would be to install two separate copies of `RexxHttp.rex`, one in a CGI-enabled directory and another one in a Mod_Rexx-enabled directory, and associate the file extensions accordingly (see *Enable RexxHttp processing*, which follows). One drawback of this method is that you end up with two copies of `RexxHttp.rex`, and that may cause maintenance problems afterwards. Another, more convenient, setup is to use the Apache `Alias` directive so that one single copy of `RexxHttp.rex` can be accessed both as a CGI *and* as a Mod_Rex `.rex` procedure. As an example to illustrate this possibility, suppose that your chosen CGI directory is `c:/server/cgi`. To allow CGI processing, you should have a

```
ScriptAlias /cgi/ "c:/server/cgi/"
```

directive in your Apache configuration file. You can then define another alias:

```
Alias /scripts/ "c:/server/cgi/"
```

Now `RexxHttp.rex` can be accessed in two ways: as `/cgi/RexxHttp.rex`, and as `/scripts/RexxHttp.rex`. If you now configure your Apache server so that `/scripts/RexxHttp.rex` gets processed by Mod_Rexx (for example, using

```
<Location /scripts>
  AddType application/x-httpd-rexx-script .rex
</Location>
```

in your Apache configuration file), then one single copy of `RexxHttp.rex` can be used for both CGI and Mod_Rexx processing. More details about this setup will be found at the end of the following subsection, in *Under both CGI and Mod_Rexx* on page 6.

2.2.3 Enable RexxHttp processing

2.2.3.1 Under CGI

For each directory in which you want to enable RexxHttp, add the following Apache directives to your Apache configuration file (of course you can also enable RexxHttp for the whole server if you so desire):

```
AddHandler rexxhttp extensions
Action rexxhttp /cgipath/RexxHttp.rex
```

Extensions is a list of blank-separated case-insensitive extensions like `.html` or `.rsp` (the leading dot is optional). For example,

```
AddHandler rexxhttp .htm .html .rsp
```

indicates that files with extensions `.htm`, `.html` and `.rsp` will be handled by RexxHttp.

Cgipath is the complete Apache path up to and including the CGI directory. For example, if you installed `RexxHttp.rex` in the standard `/cgi-bin` directory, you would have

```
Action rexxhttp /cgi-bin/RexxHttp.rex
```

Note: in the two Apache directives above, `rexxhttp` represents the name of the Apache handler. You can use any name you desire instead of `rexxhttp`, as long as you change it consistently in the two directives.

2.2.3.2 Under Mod_Rexx

For each directory in which you want to enable RexxHttp, add the following Apache directives to your Apache configuration file (of course you can also enable RexxHttp in the whole server if you so desire):

```
AddHandler rexxhttp extensions
Action rexxhttp /modrexxpath/RexxHttp.rex
```

Extensions is a list of blank-separated case-insensitive extensions like `.html` or `.rsp` (the leading dot is optional). For example,

```
AddHandler rexxhttp .htm .html .rsp
```

indicates that files with extensions `.htm`, `.html` and `.rsp` will be handled by RexxHttp.

Modrexxpath is the complete Apache path up to and including the directory where you placed `RexxHttp.rex`. For example, if you installed `RexxHttp.rex` in the `/scripts` directory, you would have

```
Action rexxhttp /scripts/RexxHttp.rex
```

Note: in the two Apache directives above, `rexxhttp` represents the name of the Apache handler. You can use any name you desire instead of `rexxhttp`, as long as you change it consistently in the two directives.

2.2.3.3 Under both CGI and Mod_Rexx

Assume that `/cgi` is a CGI directory and `/scripts` is a Mod_Rexx directory (they can be aliased to the same physical directory if you like), and assume further that both directories contain a copy of `RexxHttp.rex` (the same file if the directories have been aliased). Then you could use

```
Action rexxhttpCGI     /cgi/RexxHttp.rex
Action rexxhttpModRexx /scripts/RexxHttp.rex
AddHandler rexxhttpCGI CGI-extensions
AddHandler rexxhttpModRexx Mod_Rexx-extensions
```

to have files whose extension is in the list of *CGI-extensions* processed by RexxHttp/CGI, and files whose extension is in the list of Mod_Rexx-extensions processed by RexxHttp/Mod_Rexx.

At this point you can already test basic RexxHttp functionality. Take the `test.html` test file provided with this package, install it in some RexxHttp-enabled directory, change its extension to a valid RexxHttp extension if necessary, and try it by pointing your browser to the appropriate URL. Don't follow the links related to cookies, because there's no support for cookies yet (i.e., cookies will work, but they will have incorrect expiration dates, unless you live in the GMT timezone with no DST). All the other tests should work.

2.3 Advanced features

2.3.1 Passing variables

2.3.1.1 Under CGI

To pass variables to RexxHttp under Apache/CGI you have to use Apache's `SetEnv` directive. Some variable names are reserved by RexxHttp (for example, to indicate which page compilers to use) and will be described below. All variable names should start with an underscore character (`"_"`), and can only consist of alphabetic, numeric and underscore characters (Apache automatically translates alphabetic characters to uppercase for environment variables). You can also use `SetEnv` to set other kind of variables under Apache/CGI and access them using the `Value` builtin function, but this will make your servlets *non-portable* (see *Writing portable servlets* on 47 for more information).

Please note that `SetEnv` can be specified on a per-server config, per-virtual host, per-directory, or per-`.htaccess` file basis. You can also use the `UnsetEnv` directive to control variable passing. See your Apache documentation for details.

2.3.1.2 Under Mod_Rexx

To pass variables to RexxHttp under Apache/Mod_Rexx you have to use the `RexxSetVar` directive and Apache's `LocationMatch` directive in the way explained below. Some variable names are reserved by RexxHttp (for example, to indicate which page compilers to use) and will be described below. All variable names should start with an underscore character (`"_"`), and can only consist of uppercase alphabetic, numeric and underscore characters. When using RexxHttp under Apache/Mod_Rexx, you cannot set other types of variables, like mixed case variables or variables that do not begin with an underscore: even if you set them using the `RexxSetVar` directive, they will not be seen by your Servlet/RSP.

Per-server configuration. Use the `RexxSetVar` directive to define a special Rexx variable called `REXXVARS`, which will contain the names of all the variables passed to the Rexx servlet. The special names `_TIMEZONE`, `_FNAMETEMPLATE`, `_RSPCOMPILERS`, `_RSPCOMPILER_`*n* and `_RSPCOMPILER_`*n*`_TYPES` are automatically handled by RexxHttp and do not have to be specified manually.

The variables should be specified as follows:

```
RexxSetVar REXXVARS VAR1 VAR2 ...
RexxSetVar _VAR1 value
RexxSetVar _VAR2 value
...
```

Example:

```
RexxSetVar REXXVARS "CITIES COLOR"
RexxSetVar _CITIES "Barcelona Ibiza"
RexxSetVar _COLOR "Blue"
RexxSetVar _TIMEZONE "-03:00 00:00"
```

Per-directory configuration. Variables can be specified on a per-directory basis using the `LocationMatch` directive in the following way:

```
<LocationMatch "^/rexxhttpdir/RexxHttp.rex/dir*">
  RexxSetVar REXXVARS VAR1 VAR2 ...
  RexxSetVar _VAR1 value
  RexxSetVar _VAR2 value
  ...
```

```
</LocationMatch>
```

Rexxhttpdir is the complete Apache path to the location where `RexxHttp.rex` is installed. *Dir* is the directory where you want to assign variables to.

Example: If `RexxHttp.rex` is installed in the `/scripts` directory and you want to assign variables to the `/some/dir` directory, you would use

```
<LocationMatch "^/scripts/RexxHttp.rex/some/dir*">
  RexxSetVar REXXVARS "CITIES COLOR"
  RexxSetVar _CITIES "Barcelona Ibiza"
  RexxSetVar _COLOR "BLUE"
  RexxSetVar _TIMEZONE "-03:00 00:00"
</LocationMatch>
```

2.3.2 Customize the _TIMEZONE variable

To use Netscape-style cookies, RexxHttp needs a properly configured `_TIMEZONE` variable. The format of the `_TIMEZONE` variable is the following:

```
base dst [hour1 weekday1 pos1 month1 hour2 weekday1 pos1 month1]
```

See `SETTIMEZONE (Class method)` on page 23 for more details.

2.4 Specifying page compilers

RexxHttp can process files as Rexx Servlets, or use one or more Rexx Server Pages (RSP) compilers to compile RSP pages into Rexx Servlets and then call the compiled Servlet. By default, all extensions specified in the `AddHandler` directive are invoked as Rexx Servlets. To specify page compilers, you have to set a number of `_RSPCOMPILER`*XXXX* variables and a `_FNAMETEMPLATE` variable by using the mechanism described in the section relevant to your setup. Please note that you can change the number, nature, extension association and order of the compilers on a per-directory basis. See *Page compiler interface* on page 51 for more details.

`_FNAMETEMPLATE` is a template used to create a temporary file for the compilation phase. The value of this variable will be used to generate a temporary file name that will be passed to the page compiler. The page compiler may choose to use this file name for the compilation output, or use some other mechanism. Once the compilation phase is finished, the compiler will return indicating to RexxHttp whether there was some compilation error or whether compilation was successful, in which case it will return the name of the compiled file, which can be the temporary file, recently created, or another file (probably a cached file). RexxHttp then calls the appropiate file, and, if the temporary file was created, erases it. See *Page compiler interface* on page 51 for more details.

`_RSPCOMPILERS` should be set to a positive whole number that indicates the number of RSP compilers to use. If that number is n, for each positive whole number i between 1 and n you have to set two more variables:

`_RSPCOMPILER_`*i* is the fully qualified path and file name to a RSP compiler.

`_RSPCOMPILER_`*i*`_TYPES` is the blank-separated list of case-insensitive file extensions to associate with the i-th page compiler. A single dot (`"."`) indicates all files with no extension; in all other cases, the leading dot is optional.

Example (for Apache/CGI):

```
  setenv _rspcompilers 2
  setenv _rspcompiler_1 "c:\compilers\rspcomp.rex"
  setenv _rspcompiler_1_types "rsp"
  setenv _rspcompiler_2 "c:\compilers\rxtags11.rex"
  setenv _rspcompiler_2_types "rxt"
  setenv _fnametemplate "c:\temp\RSP???.REX"
```

Indicates that there are two RSP compilers: RSPCOMP is associated to `*.rsp` files, and RXTAGS1 to `.rxt` files. The temporary file name template is set to `"c:\temp\RSP???.REX"`.

Chapter 3

Rationale

This chapter explains why RexxHttp was written, how it was written, and why it was written the way it was written. If you are not interested in systems development, you can safely skip to the next chapter.

3.1 A very skippable historical note

One day I needed to write a CGI, so that I googled for "Rexx CGI". To my surprise, I got very few results, and they were all quite old: Les Cottrell's excellent *Guide to Writing CGI Scripts in REXX and Perl* (`http://www.slac.stanford.edu/slac/www/resource/how-to-use/cgi-rexx/`) is dated July 24, 1998; and *Writing CGI Scripts in REXX* (`http://www.webtechniques.com/archives/1996/05/rexx/`), by the same author, is dated May 1996. There's little more in the first results page (apart from an IBM page telling us to use Mod_Rexx instead, but no, thanks, I'm already using Mod_Rexx, but this time I really needed a CGI).

In particular, there was nothing about Object Rexx (or ooRexx) and CGI. How could this be? It's very easy, at least in principle, to write a nice object-oriented wrapper for the CGI specification! So that I decided to write one and make it public, so that others might benefit.

The first part was easy, even if a little tedious: I had to study the aging CGI specification (`http://hoohoo.ncsa.uiuc.edu/cgi/`) and implement wrappers for the HTTP request and response. Everything was quite straightforward, except maybe for the design of the `ARG` and `COOKIE` request methods, which I tried to keep as Rexx-like as possible. To handle the response contents, I decided to implement a subclass of Stream that would buffer its output in a MutableBuffer, so that the response could be updated at will before the contents had not been flushed; this would also have the benefit of offering increased performance.

Once I had a working prototype, I realized that I had something more than an assistant to write CGIs: this could be the core of a complete Rexx servlet system! Then an idea came to my mind: could the same system be made to run under Mod_Rexx? This would be really useful: it would allow to design *portable* *servlets*, while still getting all the benefits of Mod_Rexx or CGI if so desired. So that I put myself to work, and, *voilà*, in some hours I had a dual-pathed system that could run unchanged under Mod_Rexx and CGI. (See *Writing portable servlets* on page 47.)

As I intended to make the program public, I wanted to test it thoroughly before releasing it. I was using a notebook loaded with Apache 1.3.35/CGI and Apache 2.0.58/Mod_Rexx 2.1.0 for development and testing. I don't have access to any Unix variant, but I hoped that somebody would be kind enough to do the testing for me. Then I thought about the poor OS/2 users; I myself use WSEB 4.52/Apache 1.3.33/Mod_Rexx 1.2.0/Classic Rexx in my main servers, but I'm being slowly forced to migrate due to lack of support and development. I didn't have a working WSEB/Object Rexx system to conduct OS/2 testing (I cannot SWITCHRX my production systems: this makes them unstable when accesing DB2), so that I bought a copy of Parallels Workstation, installed a virtual WSEB 4.52, SWITCHRX'ed it, and tried RexxHttp under Apache 2.0. Then I realized that OS/2's Object Rexx is backlevel, and does not offer a MutableBuffer class – so that I decided to write a poor man's implementation of MutableBuffer, only enough of it so that RexxHttp can run.[1]

At this point I was quite excited. Could RexxHttp be made to run under some HTTP server other than Apache? After all, the CGI implementation is server-neutral. Then I realized that some of my Windows XP Professional systems included a working Microsoft IIS 5.5 server: I made some tests (I had no previous experience with IIS), and after some tweaking I got RexxHttp working. I though: if I could get the system working under the more popular servers, and since RexxHttp allows the development of portable servlets, a pool of OS-independent, server-independent, and gateway interface-independent servlets could be created, so that all (Object and oo-)Rexx users could share them.

Finally I asked myself if the current Mod_Rexx page compilers, i.e., RSPCOMP and REXXTAGS, could be run under RexxHttp: well, with a little tweaking they run flawlessly! So that I thought in more general terms and I designed a very simple compiler-neutral interface so that new compilers can be written and installed as plugins. (See *Page compiler interface* on page 51.)

That's how RexxHttp was created. The following section lists the design goals as a whole, as if all the ideas came to my mind at once.

3.2 Design goals

Design an Open Source Servlet System that uses the expresive power of ooRexx to encapsulate the complexity of the HTTP request/response model in simple to use and well documented classes that can be used as a standard. Ideally, the servlet system should be written in pure ooRexx and provide defaults in such a manner that in standard cases configuration can be kept to a minimum and deployment time nears zero. All the differences between HTTP servers, underlying gateway interfaces and operating systems should be handled by the

[1]Of course some backlevel versions of Object Rexx also lack Mutablebuffer.

servlet system, in such a way that a single source distribution can accomodate the corresponding variations between systems and implementations.

The servlet system should run under all versions of Apache, and should be able to be extended to run under other HTTP servers (for example, Microsoft IIS). The system should support the possibility of writing servlets that are independent of the underlying HTTP server.

The servlet system should run under CGI *and* under Mod_Rexx. Servlets that run under Mod_Rexx enjoy the advantages of speed and scalability under heavy loads that Mod_Rexx provides. Servlets that run under CGI are useful in cases where there is no corresponding Mod_Rexx implementation (for example, when you want to use a version of Apache for which there is no Mod_Rexx support, or if you are forced to use a server which is not Apache), and in cases where you need to use features that Mod_Rexx does not have (for example, executing system commands, or writing to the standard output stream using methods other than the SAY instruction).

The servlet system should run under Windows, all versions of Unix for which there is a corresponding Apache implementation, and under other operating systems. Special care should be taken so that the system runs under OS/2, if at all possible.

The servlet system should provide a way to write servlets that are portable without modification between all the underlying gateway interfaces (i.e., Mod_Rexx and CGI), operating systems, and HTTP servers. This is specially important for the Mod_Rexx/CGI dichotomy. Imagine the following situation: you manage a server in which part or all of the pages are Rexx Servlets and runs under Mod_Rexx. Then a new Apache version appears. You want to use this new version of Apache for whatever reason, but a Mod_Rexx implementation for this new version of Apache does not (yet) exist. If your servlets are written following the compatibility guidelines,[2] you can simply migrate to the new version of Apache (if you are willing to pay the price in performance degradation) and still run all your servlets unchanged. Once the new version of Mod_Rexx appears, you can switch back to Mod_Rexx and enjoy its enhanced performance and scalability. Notice that for personal webs and small businesses (i.e., a big percentage of the web) the performance of CGI is quite acceptable: one can easily get 20 servlet pages per second under CGI in a standard laptop loaded with several other programs, which is more than enough for a small web.

The servlet system should be able to run existing Rexx Server Pages compilers (i.e., Mod_Rexx RSPCOMP, and REXXTAGS) with minimal modification, and should provide a foundation for more sophisticated compilers that might be written in the future. Existing RSP pages should also be adaptable to run as servlets with minimal or no modification.

The servlet system should allow the simultaneous execution of rexx servlets and RSP pages, calling different page compilers based on the filetype. For example,

[2]See *Writing portable servlets* on page 47.

you could have .rex files automatically called as servlets, .rsp files be compiled with RSPCOMP, and .html files be compiled with REXXTAGS, all under the same directory.

3.3 Design decisions and compromises

3.3.1 Associating Rexx files with a Rexx handler in Apache

Apache's `Action` directive allows to define an action associated to a CGI script *or* a Mod_Rexx script:

```
  Action rexxhttp /cgi-bin/RexxHttp.rex
```

will associate the `rexxhttp` action with the `RexxHttp.rex` CGI script (assuming that, as usual, `/cgi-bin/` has been `ScriptAlias`ed), while

```
  Action rexxhttp /scripts/RexxHttp.rex
```

will associate the `rexxhttp` action with the `RexxHttp.rex` Mod_Rexx script (assuming that the `/script/` directory allows the execution of Mod_Rexx `.rex` programs).

Now we will use Apache's `AddHandler` directive to associate our newly defined handler with a set of extensions:

```
  AddHandler httprexx .htm .html .rsp
```

maps the `.htm`, `.html` and `.rsp` extensions to `rexxhttp`.

This provides a uniform approach to both CGI and Mod_Rexx. When running under Mod_Rexx, the overhead is the overhead caused by RexxHttp itself, which should not be much bigger than the overhead of using `Apache.cls`; under CGI, the little overhead of setting up the RexxHttp classes is vastly compensated by advantages of RexxHttp: having a uniform interface, clear error handling, etc..

3.3.2 Passing variables

Apache CGI environment variables are limited to letters, numbers and the underscore character; the first character cannot be a number. Mod_Rexx's `RexxSetVar` variables are more flexible in their syntax (for example, they allow dots to form stem-like variables). Although it is possible to define mixed-case CGI variables (case *is* significant under CGI), *all* standard CGI variables are passed in uppercase.

On the other hand, it is very convenient to access passed variables in a directory-like manner:

```
request~passedvariable
```

(in addition to the `request["passedvariable"]` syntax), but then a passed variable could have the same name as a Http.Request method (or a new method defined in the future). That's why the decision was made to be still more restrictive than CGI: all RexxHttp variables have to begin with an underscore, and use only uppercase letters, numbers, and underscores.

Under CGI, you can always call the `VALUE` builtin function if you need to access a variable that does not conform to these rules. Under Mod_Rexx the situation is different: although you can set as many variables as you want by using the `RexxSetVar` directive, these variables are only visible to the main program,[3] i.e., they revert to their default values as soon as a procedure is called. But using RexxHttp means that servlets should be called as external procedures from RexxHttp itself, and therefore they will not have access to the `RexxSetVar` variables. That's why the (admittedly ugly) decision to define a `REXXVARS` variable was taken: you define

```
RexxSetVar REXXVARS "var1 ... varn"
```

where *var1*, ..., *varn* are variable names, and you are supposed to define each of *_var1*, ..., *_varn* later (note the underscores). For example,

```
RexxSetVar REXXVARS "COLOR CITIES"
RexxSetVar _COLOR BLUE
RexxSetVar _CITIES "Barcelona Ibiza"
```

RexxHttp then fetches the `REXXVARS` variable, parses it, fetches each *_var*i variable, and stores everything in a directory so that it can be accessed later.

Since some variables are defined by RexxHttp itself (i.e., `_TIMEZONE`, all the `_RSPCOMPILER`*xxxx* variables, and `_FNAMETEMPLATE`), RexxHttp automatically appends these variables to `RexxVars` when running under Mod_Rexx.

Under Apache/CGI, variables can be set by using the `SetEnv` and related directives. Apache treats the `Action`/`SetHandler` combination as a redirection, and passes the per-directory (or `Location`, etc.) variables in a new set of `REDIRECT_`*xxxx* variables. HttpRexx takes care of that under Apache/CGI: if a request for a previously unfetched variable `_var` is made, first an attempt is made to fetch `REDIRECT__var` using `VALUE`, and, only if this fails (i.e., the null

[3]This was not so for Mod_Rexx 1.2.0 under OS/2. Was this an OS/2 feature?

string is returned) an attempt is made to fetch `_var`. In any case, the result is cached so that as little calls to `VALUE` as possible are made.

Under Apache/Mod_Rexx, per-server variables are passed without problems, but per-directory variables are not passed (i.e., you cannot expect to `RexxSetVar` a variable V for directory `/test`, then request `/test/sample.rex`, and get V passed to `sample.rex`, because Mod_Rexx thinks that `RexxHttp.rex` is being called instead). To overcome this problem, you can use the following procedure: assume that `RexxHttp.rex` is installed in directory `/scripts`; when requesting a `file` in directory `/test`, Apache will call `RexxHttp.rex`, *but with a composite URL of*

```
/scripts/RexxHttp.rex/test/file
```

and this allows us to use the `LocationMatch` Apache directive to set the variables:

```
<LocationMatch "^/scripts/RexxHttp.rex/test/*">
  RexxSetVar REXXVARS "COLOR"
  RexxSetVar _COLOR "CYAN"
</LocationMatch>
```

Admittedly, this is not very elegant, but it works as intended.[4]

3.3.3 The problem of request values

CGI defines a set of environment variables to pass information about the request to the CGI program. Mod_Rexx defines a set of `WWW`*xxxx* variables which provide *similar* information to the Mod_Rexx program. A Rexx servlet should expect to receive a set of values, accesible using the request object' methods. But here we have two problems: 1) the information passed to both CGI and Mod_Rexx is *not* the information that a servlet would expect to receive: this is because what is really being called is `HttpRexx.rex`, not the servlet; and 2) the information passed to CGI and Mod_Rexx is not consistent (because the CGI specification is ambiguous or inconsistent, or because the values of the variables are not what one should expect, or because CGI and Mod_Rexx return different values).

The first problem can be solved easily, because, even if the servlet engine does not receive a set of variables which are ready to be passed to the servlet, those values can be computed quite easily from the received values (i.e., we get *enough information* to reconstruct a suitable set of CGI-like variables).

The second problem cannot be solved without taking a number of design decisions:

[4] If somebody knows of a simpler solution, please email me.

3.3.3.1 `PATH_INFO`, `PATH_TRANSLATED`, `SCRIPT_FILENAME` and `WWWFILENAME`

The `PATH_TRANSLATED` variable has an inconsistent definition in the CGI specification:[5] on the one hand, `PATH_INFO` is defined as "The extra path information, as given by the client." Then the CGI specification goes on to define `PATH_TRANSLATED` as follows: "The server provides a translated version of `PATH_INFO`, which takes the path and does any virtual-to-physical mapping to it." Of course this is nonsense: if a URL of the form

```
http://somehost/page.html/extra/path/info
```

is used, `PATH_INFO` is `"/extra/path/info"`, and then `PATH_TRANSLATED` should be "a translated version of `PATH_INFO`" (translated to what?), "which takes the path" (which path? `PATH_INFO` itself?) "and does any virtual-to-physical mapping to it" – but `PATH_INFO`, in general, is designed *not* to have any physical mapping!

Apache, for example, does *not* set `PATH_TRANSLATED` when calling a CGI, *but*, if the CGI has some extra path information, it returns the real path info *without the filename*, but appended to the directory where the CGI resides — a useless value.[6] Similarly, Mod_Rexx defines `PATH_TRANSLATED` to be "the fully qualified path and filename of the script", but, since it returns the values provided by Apache, it gives the wrong result when there is some extra path info (but the right result when there is not, unlike CGI). RexxHttp's `PATH_TRANSLATED` method consistently returns the fully qualified path *and* filename of the called script, regardless of whether there is some extra path info or not. And, again consistently, `PATH_INFO` returns the (translated) path info when there is some path info, and the null string when there is no path info.

3.3.3.2 `SCRIPT_NAME`

`SCRIPT_NAME` is defined by the CGI specification as "A virtual path to the script being executed, used for self-referencing URLs", and Mod_Rexx defines `WWWSCRIPT_NAME` as "the fully qualified URI path and name of the script". This seems to imply that no extra path information should be included in the value of this variable. Then Apache returns the virtual path with no extra path info, and Mod_Rexx returns the virtual path *with* the (translated) extra path info, if there is such path info. RexxHttp consistently returns the virtual path to the servlet or RSP file in all cases (that is, without any extra path info).

3.3.3.3 `REQUEST_URI` and `WWWUNPARSEDURI`

Apache defines a non-standard CGI variable called `REQUEST_URI` as "the portion of the URL following the scheme and host portion". This is equivalent to Mod_Rexx `WWWUNPARSEDURI`, which is defined (somewhat strangely)

[5] See `http://hoohoo.ncsa.uiuc.edu/cgi/env.html`.

[6] Apache returns what should be `PATH_TRANSLATED` in the non-standard CGI variable `SCRIPT_FILENAME`, but with all backslashes translated to forward slashes. This is the same as Mod_Rexx `WWWFILENAME`. RexxHttp has a `FILENAME` method which returns the same value, but with the backward slashes intact, if any.

as "the unparsed portion of the request URI". RexxHttp defines an equivalent `REQUEST_URI` method and an alias `UNPARSEDURI`. Then Mod_Rexx defines `WWWURI` as "the entire request URI", and returns the translated URI minus the query string, if present. This is the value returned by RexxHttp `URI` method.

3.3.3.4 Other differences between Apache/CGI and /Mod_Rexx

`DOCUMENT_ROOT`, `REMOTE_PORT`, `SERVER_ADDR` and `SERVER_SIGNATURE` are specific to Apache/CGI and do not exist under Mod_Rexx. These methods always return the null string under Mod_Rexx.

3.3.4 Netscape-style cookies and timezones

Rexx does not have any builtin, operating-system independent way of knowing in which timezone it is operating. Netscape-style cookies use an *expires=date* attribute-value pair to indicate their expiration date.[7] This has several problems: 1) The time at the server should be right (this is not a problem with Windows, which can be set to sinchronize with a time server, but can be a problem in other operating systems, for example vanilla WSEB, which does not have any kind of synchronization, not even automatic Daylight Savings Time (DST) switching); 2) The time at the client should be right *and coincide with the time at the server*; this can pose security risks if for some reason the time at the client is earlier than the time at the server; 3) the *expires* attribute should be the time *in the GMT timezone*. This is the real problem: since Rexx does not know in which timezone it is operating, it cannot reliably calculate the current time in the GMT timezone.

Fortunately, most timezones work in the same way: there is a standard offset from GMT, and an extra offset to apply when DST is active. DST works *for most timezones* as follows: DST is activated or deactivated at a specified hour (normally in the early morning) on a certain weekday of a specified month, for example, at 2am in the first friday of october, or at 4am in the last sunday of september. Timezone handling is implemented by setting the `_TIMEZONE` variable (see *Customize the _TIMEZONE variable* on page 8 and *SETTIMEZONE (Class method)* on page 23 for details).

[7] RFC2109-style cookies do not exhibit this problem, since they make use of the `Max-age` attribute. Unfortunately, browser support for RFC2109 is erratic.

Chapter 4

Tutorial

4.1 Introduction

A *Rexx servlet* is a Rexx external procedure residing in a HTTP server (for example, Apache, or Microsoft IIS) and designed to programmatically produce the contents of a server page (the *response*), or, more generally, a server object (for example, a graphics file, a pdf object, or a mp3 stream), after some agent (the *requester*: normally a user's browser) has asked for it.

A Rexx servlet receives two arguments: the *request*, an object that encapsulates the details about the HTTP request and assists the servlet programmer in accesing its values in a structured manner, and the *response*, an object that helps in assembling and sending the resulting server object (i.e. the HTTP headers and the response contents) back to the requester.

The RexxHttp servlet engine can process Rexx servlets and RSP pages written to run under Mod_Rexx or CGI under different HTTP servers (currrently, Apache and some versions of Microsoft IIS[1]). Servlets and RSP pages can be written in a *portable* way; see Writing portable servlets in page 47 for details. In the rest of this chapter we will show how to write both portable and non-portable servlets.

4.2 A "Hello, world" example

Let us start first with a minimal (CGI-only) servlet:

```
say "Hello, world!"
```

This is indeed minimal! Under CGI, the servlet engine automatically redirects `SAY` output to a buffered Http.OutputStream; once the servlet ends, the response is committed, and then flushed. This ensures that the (default) HTTP headers are actually emitted *before* the contents of the page.

This program is included in the RexxHttp package as `sample1.smp` on the `/samples/cgionly` directory.

[1]IIS support is experimental in this release.

4.3 A portable "Hello, world" example

The previous example works only under CGI. Now we want to write a servlet which works *both* under CGI *and* Mod_Rexx:

```
1: use arg request, response
2: output = response~output
3: response["Content-Type"] = "text/plain"
4: output~say("Hello, world!")
```

The first two lines are standard for all servlets. Line 1 retrieves the Http.Request and Http.Response objects that RexxHttp automatically creates; line 2 retrieves the servlet output stream object. Line 3 sets the response type, and line 4 outputs a single line.

This program is included in the RexxHttp package as `sample2.smp` on the `/samples/portable` directory.

4.4 Output buffering and alternative syntax

Let us now consider the following servlet, which is equivalent to the previous one (that is, it produces the same output):

```
1: use arg request, response
2: output = response~output
3: output~say("Hello, world!")
4: response~Content_Type = "text/plain"
```

Two things are to note: 1) Line 3 writes some output *before* setting the response type: this is possible because standard output is redirected to a buffer, which is only flushed automatically when the servlet ends or the buffer grows to be bigger than a maximum size.[2] 2) The response content type is set using an alternative syntax: this true in general: all HTTP headers can be set using

`response["`*header-name*`"] = `*value*

or using

`response~`*header_name* `=` *value*,

where *header_name* is the same as *header-name*, but changing all occurrences of `"-"` by `"_"`.

This program is included in the RexxHttp package as `sample3.smp` on the `/samples/portable` directory.

[2]No maximum is implemented at this time.

4.5 Retrieving GET/POST arguments

The following RexxHttp/CGI program lists all the GET/POST arguments and their values. Since we want to keep things simple, we are producing plain text output so that the program structure is not cluttered by HTML logic.

```
use arg request
say "List of passed parameters:"
say "+"||"-"~copies(20)"+"||"-"~copies(20)"+"
do i = 1 To request~arg()
  say "|" || request~arg(i,"Name")~left(20) || "|" ||,
      request~arg(i,"Value")~left(20) || "|"
end
say "+"||"-"~copies(20)"+"||"-"~copies(20)"+"
say "Total:" request~arg() "parameters"
```

If you request this servlet with a query string of `a=b&c=d`, the servlet will output

```
List of passed parameters:
+--------------------+--------------------+
|a                   |b                   |
|c                   |d                   |
+--------------------+--------------------+
Total: 2 parameters
```

This program is included in the RexxHttp package as `sample4.smp` on the `/samples/cgionly` directory.

4.6 Generating PDF output

We are now ready for a more sophisticated example. Assume that you want to generate PDF output. We will assume that you have LaTeX installed, and we will use `pdflatex`. Since Mod_Rexx does not allow the execution of commands, this servlet will be a CGI servlet.

```
use arg request, response
dir = "C:\rexxhttp\temp\"
call directory dir
fn = SysTempFilename(dir"XXX?????.tex")
call lineout fn,"\documentclass[a4paper]{article}"
call lineout fn,"\begin{document}"
if request~arg("name","Omitted") then
     call lineout fn,"Hello, no name!"
else call lineout fn,"Hello," request~arg("name")"!"
call lineout fn,"\end{document}"
call lineout fn
"pdflatex" fn "-interaction=batchmode > nul"
f = fn~substr(1,fn~length - 3)
pdfn = f"pdf"
```

```
15: size = stream(pdfn,"c","query size")
16: response~content_type="application/pdf"
17: pdfContents = charin(pdfn,,size)
18: response~output~charout(pdfContents)
19: call stream pdfn,"c","close"
20: call SysFileDelete f"tex"; call SysFileDelete f"pdf"
21: call SysFileDelete f"aux"; call SysFileDelete f"log"
```

We do not do any kind of error handling to keep the program length to a minimum. The program takes a query argument called `"name"` (the string `"no name"` is substituted if no such argument exists), dynamically creates a temporary `.tex` file with a customized greeting, calls `pdflatex`, returns the resulting `.pdf` file as the response, and erases all temporary files. The program has been tested under Windows XP with the MiKTeX distribution of (La)TeX. You might need to change the temporary directory in line 2.

This program is included in the RexxHttp package as `sample5.smp` on the `/samples/cgionly` directory.

Chapter 5

Class Reference

5.1 The Http.Cookie class

A Http.Cookie object is used to store HTTP cookies. Support is provided for Netscape-style[1] ("version 0") cookies and RFC2109[2] ("version 1") cookies. Please note that support for RFC2109 cookies is inconsistent amongst different browsers.

5.1.1 SETTIMEZONE (Class method)

```
>>-SETTIMEZONE(timezone_string)--------------------------------><
```

Sets the timezone information to be used when constructing the `Expires` attribute of Nescape-style cookies. The *timezone_string* should have the following format:

```
base dst [hour1 weekday1 pos1 month1 hour2 weekday1 pos1 month1]
```

where:

Base is the (eventually signed) base offset from GMT to apply when DST is not in effect, in the format `[+|-]hh[:mm[:ss]]`, i.e., when DST does not apply, *GMT time + base = local time.*

Dst is the (eventually signed) time offset that should be applied to *base* when DST applies, in the format `[+|-]hh[:mm[:ss]]`, i.e., if DST is in effect, then *GMT time + base + dst = local time.* If your timezone does not use DST, set DST to `0` (or `00:00`, etc.). The following tokens indicate the starting and ending datetimes for DST, and are not used when DST = 0.

The quadruple *hour1 weekday1 pos1 month1* is interpreted in the following way: *hour1* is the (unsigned) time of the day when DST begins, in the format `hh[:mm[:ss]]`. DST will begin on the *pos1*-th day of the week corresponding to the whole number *weekday1* (where 1=Mon,...,7=Sun) on month *month1*

[1] See `http://wp.netscape.com/newsref/std/cookie_spec.html`.
[2] See `http://www.ietf.org/rfc/rfc2109.txt`.

(where 1=Jan,...,12=Dec). If *pos1 = 5* this means the last *weekday1*-th day of the week of the month.

The quadruple *hour2 weekday2 pos2 month2* is interpreted in the same way, and designates the datetime when DST ends.

Examples:

```
.Http.Cookie~setTimeZone("+01:00 +01:00 02:00 5 1 10 03:00 7 5 4")
```

The base offset from GMT is +1 hour when DST is not in effect, and +2 hours when DST is in effect. DST starts at 2 am on the first (1) Friday (5) of October (10), and ends at 3 am of the last (5) Sunday (7) of April (4).

```
.Http.Cookie~setTimeZone("-06:00 00:00")
```

The base offset from GMT is -6 hours. There is no DST.

Note: This class method is automatically called by RexxHttp at startup. RexxHttp evaluates the `_TIMEZONE` variable, and passes its unchanged value to the method. If there is some error in the `_TIMEZONE` variable, a `SYNTAX` error is raised; is the `_TIMEZONE` variable is not set, the GMT timezone is assumed, with no DST.

5.1.2 TIMEZONEOFFSET (Class method)

```
>>-TIMEZONEOFFSET---------------------------------------------><
```

Returns a whole number expressing the current timezone offset from GMT in seconds. The returned value takes into account DST variations. The value returned is such that *local time - timezoneoffset = GMT time.*

Examples:

```
-- After...
.Http.Cookie~setTimeZone("+01:00 +01:00 02:00 5 1 10 03:00 7 5 4")

-- On 20062109, at any time,
.Http.Cookie~TimeZoneOffset  -> 3600     -- 1 * 60 * 60

-- The first Friday in October 2006 is 6.

-- On 20061006 at 1:15:00,
.Http.Cookie~TimeZoneOffset  -> 3600     -- 1 * 60 * 60

-- On 20061006 at 2:00:00,
.Http.Cookie~TimeZoneOffset  -> 7200     -- 2 * 60 * 60

-- After .Http.Cookie~setTimeZone("-06:00 00:00") ...
.Http.Cookie~TimeZoneOffset  -> -21600   -- -6 * 60 * 60
                                         -- (always: no DST)
```

5.1.3 INIT

```
>>-INIT(name,value)------------------------------------------><
```

Initializes a new cookie object. The *name* and the *value* of the cookie are strings. The name must conform to RFC2109, that is, it must only use ASCII non-control characters different from `()<>@,;:\"/[]?={}‘`, and cannot begin with `"$"`. The value can be any string. Take into account that the cookie name plus the cookie value cannot exceed 4096 bytes, and that non-ASCII characters and some other characters will have to be encoded before storing the cookie value. A safe assumption would be that the value can use at least 1000 characters.

5.1.4 COMMENT

```
>>-COMMENT---------------------------------------------------><
```

Returns a string containing the comment of the cookie, or the null string if the comment is not set (the default).

5.1.5 COMMENT=

```
>>-COMMENT=comment-------------------------------------------><
```

Sets the comment of the cookie to *comment*, which must be a string. If *comment* is equal to the null string, the comment is unset.

5.1.6 DOMAIN

```
>>-DOMAIN----------------------------------------------------><
```

Returns a string containing the domain of the cookie, or the null string if the domain is not set.

5.1.7 DOMAIN=

```
>>-DOMAIN=domain---------------------------------------------><
```

Sets the domain of the cookie to *domain*, which must be a string. If *domain* is equal to the null string, the domain is unset.

5.1.8 MAKESTRING

```
>>-MAKESTRING------------------------------------------------><
```

Returns a string representation of the cookie. The returned string is suitable to be the value of a Set-Cookie header.

Example:

```
cookie~makestring
       -> "NAME=value; expires=Tue, 10-Oct-2006 14:26:28 GMT"
```

5.1.9 MAX_AGE

```
>>-MAX_AGE-----------------------------------------------------><
```

Returns the maximum age of the cookie as a whole number. If none was specified, a negative value is returned. See MAX_AGE=, which follows.

Example:

```
cookie~max_age  ->  120 /* Cookie lasts 2 minutes */
```

5.1.10 MAX_AGE=

```
>>-MAX_AGE=seconds---------------------------------------------><
```

Sets the maximum age of the cookie to *seconds*, which must be a whole number. The cookie will expire after *seconds* seconds have passed. If *seconds* is negative, the cookie is a *session cookie* (the default) and will be deleted when the requester (usually the web browser) exits. If *seconds* is zero, the cookie will be deleted.

Example:

```
cookie~max_age = -3  ->  /* Cookie is now a session cookie */
```

5.1.11 NAME

```
>>-NAME--------------------------------------------------------><
```

Returns a string containing the name of the cookie.

Example:

```
cookie~name     ->  "MYCOOKIE"   /* Maybe */
```

5.1.12 PATH

```
>>-PATH--------------------------------------------------------><
```

Returns a string containing the path of the cookie, or the null string if the path has not been set (the default).

5.1.13 PATH=

```
>>-PATH=path---------------------------------------------------><
```

Sets the path of the cookie to *path*. If *path* is the null string, the path is unset.

5.1.14 SECURE

```
>>-SECURE------------------------------------------------------><
```

Returns 1 if the cookie is secure, or 0 if the cookie is not secure (the default).

5.1.15 SECURE=

```
>>-SECURE=value------------------------------------------------><
```

Sets the secure attribute of the cookie to *value*, which must be 0 or 1.

5.1.16 VALUE

```
>>-VALUE-------------------------------------------------------><
```

Returns the value of the cookie.

5.1.17 VALUE=

```
>>-VALUE=value-------------------------------------------------><
```

Sets the value of the cookie to *value*.

5.1.18 VERSION

```
>>-VERSION-----------------------------------------------------><
```

Returns the version number of the cookie, i.e., 0 for Netscape-style cookies, and 1 for RFC2109-style cookies.

5.1.19 VERSION=

```
>>-VERSION=version---------------------------------------------><
```

Sets the version number of the cookie to *version*, which must be a positive whole number between 0 and 1. If *version* is 0, this is a Netscape-style cookie; if *version* is 1, this is a RFC2109 cookie.

5.2 The Http.OutputStream class

Http.OutputStream is a subclass of the builtin Stream class that stores its output in an internal buffer and has an associated Http.Response object. Each Http.OutputStream has an underlying stream where the output contents is written when the stream is closed or flushed. At creation time, the stream is marked as transient and is open for output only, unless the underlying stream is not in the `READY` state, in which case the stream remains closed and cannot be opened. All the Stream input methods will raise the `NOTREADY` condition and return a default value. Closing the stream has no effect other than flushing it (i.e., it remains open after a `CLOSE` method call).

The following methods are input methods and should not be called

```
ARRAYIN
CHARIN
CHARS
LINEIN
LINES
MAKEARRAY
SUPPLIER
```

The following methods are overriden from their corresponding Stream methods without changing their semantics. See the documentation for the builtin Stream class for details.

```
ARRAYOUT
CHAROUT
COMMAND
DESCRIPTION
LINEOUT
POSITION
QUERY
SAY
SEEK
STATE
UNINIT
```

5.2.1 INIT

```
>>-INIT(underlying_stream,request,response)---------------------><
```

Initialize the stream by creating the internal buffer, setting the *underlying_stream*, and storing the *request* and the *response*. The stream name is set to `"HTTPOUT:1"` if the qualified name of the previous stream does not begin by `"HTTPOUT:n"`, where n is a positive whole number; otherwise, a stream name of the form `"HTTPOUT:m"` is assigned, where $m = n + 1$. The stream timestamp is set to the current timestamp.

5.2.2 CLOSE

```
>>-CLOSE-------------------------------------------------------><
```

If the file is opened, flushes the stream and returns `"READY"`. The stream remains opened. See `FLUSH`, which follows, for more information.

5.2.3 FLUSH

```
>>-FLUSH-------------------------------------------------------><
```

If the associated response is not commited, this method first commits the response (see the `COMMIT` method of the Http.Response class on page 43 for more details). Then the contents of the buffer are written to the underlying stream and the buffer is reset. Finally, the underlying stream is also flushed, and `"READY:"` is returned.

5.2.4 OPEN

```
>>-OPEN--------------------------------------------------------><
```

Returns the state of the stream and does nothing else. See the description in *The Http.OutputStream class* on page 28 for more information.

5.2.5 QUALIFY

```
>>-QUALIFY-----------------------------------------------------><
```

Returns the stream name, which has always the form `"HTTPOUT:n"`. See `INIT` on page 28.

5.2.6 UNDERLYINGSTREAM

```
>>-UNDERLYINGSTREAM--------------------------------------------><
```

Returns the underlying stream.

5.3 The Http.Request class

Objects of the Http.Request class provide a convenient abstraction that encapsulates in a server-independent, OS-independent, and gateway interface-independent way the details of an HTTP request. RexxHttp automatically creates a Http.Request object and passes it to the Rexx servlet for processing. A Http.Request object has methods to access standard CGI variables (both under CGI and Mod_Rexx), GET/POST arguments, HTTP cookies, and user-defined variables.

5.3.1 INIT

```
>>-INIT(servlet_processor,mod_rexx_vars)-----------------------><
```

Initialize the request. *Servlet_processor* is the fully qualified stream name of the `RexxHttp.rex` servlet processor. *Mod_rexx_vars* is either the NIL object when running under CGI, or a directory containing Mod_Rexx's `WWW`*xxxx* variables, environment variables and the request pointer. RexxHttp automatically sets these values when creating the request object.

5.3.2 []

```
>>-[method_name]-----------------------------------------------><
```

Allows for an alternative, directory-like, syntax for the Http.Request class. The *entry_name* is uppercased, and all occurrences of `"-"` are translated to `"_"`; if the resulting string is the name of a known Http.Request method (other than `"[]"` itself), the corresponding method is called with an empty argument list; otherwise, the message is forwarded to the `UNKNOWN` method. See `UNKNOWN` on page 40 for more details.

Examples:

```
request["Content-Type"]      -- Same as request~content_type
request["HTTP-ACCEPT"]       -- Same as request~http_accept
request["_FNameTemplate"]    -- Same as request~_fnametemplate
```

5.3.3 _variable

```
>>-_variable_name----------------------------------------------><
```

Returns the passed variable *_variable_name*. This method is handled by the `UNKNOWN` method. Set `UNKNOWN` on page 40 for more details.

Examples:

```
request~_RSPCompilers      -> 2                 /* Maybe */
request~_Cities            -> "Ibiza Barcelona" /* Maybe */
```

5.3.4 ARG

```
>>-ARG--+----------------------+-----------------------------><
        +-(n--+---------+-)-----+
        |     +-,option-+       |
        +-(string-+---------+-)-+
                  +-,option-+
```

Returns information about the GET/POST arguments passed to the program.

If you specify neither *n* nor *name*, the number of arguments is returned. Note that duplicate argument names count as different arguments.

If you specify only *n*, the *n*-th argument string is returned. If *n* is greater than the number of arguments, the null string is returned. *N* must be a positive whole number.

If you specify only *string*, the value of the argument with name *string* is returned, if one exists. Otherwise the null string is returned. *String* cannot be a positive whole number.

If you specify *option*, the value returned depends on the value of *option*. The following are valid options. (Only the capitalized letter is needed; all characters following it are ignored.)

Exists If *n* was specified, returns `1` if the *n*-th argument exists (i.e., if there are at least *n* arguments); otherwise, it returns `0`. If *string* was specified, returns `1` if an argument named *string* exists; otherwise, it returns `0`.

Name If *n* was specified, returns the name of the *n*-th argument, if there are at least *n* arguments; otherwise, it returns the null string. If *string* was specified, returns *string* if an argument named *string* exists; otherwise, it returns the null string.

Omitted If *n* was specified, returns `1` if there are less than *n* arguments; otherwise it returns `0`. If *string* was specified, returns `1` if there is no argument named *string*; otherwise it returns `0`.

Value If *n* was specified, returns the value of the *n*-th argument, if there are at least *n* arguments; otherwise, it returns the null string. If *string* was specified, returns the value of the last argument with name *string*, if such an argument exists; otherwise, it returns the null string.

Note: When the request method is GET, query parameters are parsed from left to right. Therefore, name-based `ARG` method calls will always refer to the last occurrence of a parameter. When the request method is POST, no special order of the parameters should be assumed.

Note: If running under Mod_Rexx and the request method is POST, the HTTP response has to be committed; otherwise, an internal error will occur.

Examples:

```
/* Following GET /somedir/program.rex?a=b&c=23&d=&a=13 */

request~arg()             -> 4    /* Not 3   */
request~arg(1,"V")        -> "b"
request~arg(2,"Name")     -> "c"
request~arg("d","exist")  -> 1
request~arg("a","Value")  -> 13   /* Not "b" */
request~arg("k","Omit")   -> 1
```

5.3.5 AUTH_TYPE

```
>>-AUTH_TYPE---------------------------------------------------><
```

Returns the protocol-specific authentication method used to validate the user, if the server supports authentication and the script is protected. Otherwise the method returns the null string.

Examples:

```
request~auth_type         -> ""       /* Script not protected */
request~auth_type         -> "BASIC"  /* Maybe                */
```

Note: The value returned by this method is the value of the standard CGI environment variable AUTH_TYPE. Under Mod_Rexx the WWWAUTH_TYPE variable has the same value.

5.3.6 CONTENT_LENGTH

```
>>-CONTENT_LENGTH----------------------------------------------><
```

For queries which have attached information, such as HTTP POST and PUT, returns the length of the data. Otherwise the method returns the null string.

Note: The value returned by this method is the value of the standard CGI environment variable CONTENT_LENGTH. Under Mod_Rexx, the WWWCONTENT_LENGTH variable has the same value.

5.3.7 CONTENT_TYPE

```
>>-CONTENT_TYPE------------------------------------------------><
```

For queries which have attached information, such as HTTP POST and PUT, returns the content type of the data. Otherwise the method returns the null string.

Note: The value returned by this method is the value of the standard CGI environment variable CONTENT_TYPE. Under Mod_Rexx, the WWWCONTENT_TYPE variable has the same value.

5.3.8 COOKIE

```
>>-COOKIE-+-----------------------+--------------------------><
          +-(n--+---------+-)-----+
          |     +-,option-+       |
          +-(string-+---------+-)-+
                    +-,option-+
```

Returns information about the HTTP cookies passed to the program.

If you specify neither *n* nor *name*, the number of cookies is returned. Note that duplicate cookie names count as different cookies.

If you specify only *n*, the value of *n*-th cookie is returned. If *n* is greater than the number of cookies, the null string is returned. *N* must be a positive whole number.

If you specify only *string*, the value of the cookie named *string* is returned, if one exists. Otherwise the null string is returned. *String* cannot be a positive whole number.

If you specify *option*, the value returned depends on the value of *option*. The following are valid options. (Only the capitalized letter is needed; all characters following it are ignored.)

Exists If *n* was specified, returns `1` if the *n*-th cookie exists (i.e., if there are at least *n* cookies); otherwise, it returns `0`. If *string* was specified, returns `1` if a cookie named *string* exists; otherwise, it returns `0`.

Name If *n* was specified, returns the name of the *n*-th cookie, if there are at least *n* cookies; otherwise, it returns the null string. If *string* was specified, returns *string* if a cookie named *string* exists; otherwise, it returns the null string.

Omitted If *n* was specified, returns `1` if there are less than *n* cookies; otherwise it returns `0`. If *string* was specified, returns `1` if there is no cookie named *string*; otherwise it returns `0`.

Value If *n* was specified, returns the value of the *n*-th cookie, if there are at least *n* cookies; otherwise, it returns the null string. If *string* was specified, returns the value of the last cookie with name *string*, if such an argument exists; otherwise, it returns the null string.

Examples:

```
request~cookie()          -> 3    /* Maybe   */
request~cookie(1,"V       -> "b"  /* Maybe   */
request~cookie(2,"Name")  -> "a"  /* Maybe   */
request~cookie(4,"exist") -> 0
```

5.3.9 DOCUMENT_ROOT

```
>>-DOCUMENT_ROOT-----------------------------------------------><
```

If running under Apache/CGI, returns the value of the DocumentRoot Apache configuration directive. Otherwise the null string is returned.

Examples:

```
request~document_root  -> "c:/server" /* Maybe */
request~document_root  -> ""   /* If running under Mod_Rexx */
```

5.3.10 FILENAME

```
>>-FILENAME----------------------------------------------------><
```

Returns the fully qualified path and filename of the CGI or RSP file.

Example:

```
request~filename  -> "c:\test\test.html" /* Maybe */
```

Note: In Mod_Rexx the `WWWFILENAME` variable has a similar value. However, the Mod_Rexx variable has backslashes (`"\"`) translated to forward slashes (`"/"`) under Windows and OS/2. The value returned by the `FILENAME` method is ready to be used as a stream name.

5.3.11 GATEWAY_INTERFACE

```
>>-GATEWAY_INTERFACE-------------------------------------------><
```

Returns a string containing the name of the underlying gateway interface. When running under CGI, the returned string is normally `"CGI/1,1"`; when running under Mod_Rexx, this is the value of the `WWWGATEWAY_INTERFACE` variable.

Examples:

```
request~gateway_interface  -> "CGI/1.1"        /* For CGI      */
request~gateway_interface  -> "Mod_Rexx/2.1.0" /* For Mod_Rexx */
```

5.3.12 HTTP_xxxx

```
>>-HTTP_header-------------------------------------------------><
```

Returns a string containing the value of the `HTTP_`*xxxx* header. All HTTP header methods calls are handled by the `UNKNOWN method`. See `UNKNOWN` on page 40 for more details.

Examples:

```
request~http_accept_charset  -> "ISO-8859-1,utf-8;q=0.7,*;q=0.7"

-- The following is an equivalent method call
request["HTTP-ACCEPT-CHARSET"]
```

5.3.13 METHOD

```
>>-METHOD-----------------------------------------------------><
```

This method is the same that `REQUEST_METHOD`. See `REQUEST_METHOD` on page 37.

5.3.14 MOD_REXX

```
>>-MOD_REXX---------------------------------------------------><
```

Returns `1` if running under Mod_Rexx, and `0` otherwise.

5.3.15 PATH_INFO

```
>>-PATH_INFO--------------------------------------------------><
```

Returns the extra path information, as given by the client, or the null string if no path info is present.

Here are some examples:

```
-- After GET /somedir/somefile.html/extra/path/info
request~path_info  -> "/extra/path/info"

-- After GET /somedir/somefile.html
request~path_info  -> ""
```

Note: The value returned by this method is the value of the standard CGI environment variable `PATH_INFO`. Under Mod_Rexx the `WWWPATH_INFO` variable has the same value.

5.3.16 PATH_TRANSLATED

```
>>-PATH_TRANSLATED--------------------------------------------><
```

Returns the fully qualified path and filename of the script.

Note: Under Mod_Rexx the `WWWPATH_TRANSLATED` variable has a similar value. However, the Mod_Rexx variable has backslashes (`"\"`) translated to forward slashes (`"/"`) under Windows and OS/2, and does not include the filename when there is some path info. The value returned by the `PATH_TRANSLATED` method contains the filename, and is ready to be used as a stream name. The `FILENAME` method returns the same value (see `FILENAME` on page 34).

5.3.17 POST_STRING

```
>>-POST_STRING------------------------------------------------><
```

If the request method is "POST" and the arguments have been processed, this method returns the unparsed name/value pairs of the POST arguments sent to the server; in all other cases, the null string is returned.

Note: Under Mod_Rexx the WWWPOST_STRING variable has the same value.

Note: To ensure that the arguments have been processed, issue any valid call to the ARG method. See ARG on page 31 for more details.

5.3.18 QUERY_STRING

```
>>-QUERY_STRING--------------------------------------------------><
```

If the request method is "GET", this method returns the unparsed query string part of the request URI; if no query string was specified, it returns the null string.

Note: The value returned by this method is the value of the standard CGI environment variable QUERY_STRING. Under Mod_Rexx the WWWQUERY_STRING variable has the same value.

5.3.19 REMOTE_ADDR

```
>>-REMOTE_ADDR---------------------------------------------------><
```

Returns the IP address of the remote host making the request.

Note: The value returned by this method is the value of the standard CGI environment variable REMOTE_ADDR. Under Mod_Rexx the WWWREMOTE_ADDR variable has the same value.

5.3.20 REMOTE_HOST

```
>>-REMOTE_HOST---------------------------------------------------><
```

Returns the hostname of the remote host making the request, if it is known; otherwise it returns the null string.

Note: The value returned by this method is the value of the standard CGI environment variable REMOTE_HOST. In Mod_Rexx the WWWREMOTE_HOST variable has the same value.

5.3.21 REMOTE_IDENT

```
>>-REMOTE_IDENT--------------------------------------------------><
```

If the HTTP server supports RFC 931 identification, then this method returns a string containing the remote user name retrieved from the server. In all other cases it returns the null string.

Note: The value returned by this method is the value of the standard CGI environment variable REMOTE_IDENT. Under Mod_Rexx the WWWREMOTE_IDENT variable has the same value.

5.3.22 REMOTE_PORT

```
>>-REMOTE_PORT--------------------------------------------------><
```

Returns a string containing the port being used by the remote user, if known; otherwise, the null string is returned.

5.3.23 REMOTE_USER

```
>>-REMOTE_USER--------------------------------------------------><
```

If the server supports user authentication and the script is protected, returns a string containing the authenticated username; otherwise, the null string is returned.

Note: The value returned by this method is the value of the standard CGI environment variable `REMOTE_USER`. Under Mod_Rexx the `WWWREMOTE_USER` variable has the same value.

5.3.24 REQUEST_METHOD

```
>>-REQUEST_METHOD-----------------------------------------------><
```

Returns a string containing the request method used to make the request, i.e., `"GET"`, `"POST"`, `"HEAD"`, etc..

Examples:

```
request~method  -> "GET"    /* Maybe */
request~method  -> "POST"   /* Maybe */
```

Note: The value returned by this method is the value of the standard CGI environment variable `REQUEST_METHOD`. Under Mod_Rexx the `WWWREQUEST_METHOD` variable has the same value. Notice also that Mod_Rexx handles `HEAD` requests automatically, without passing them to the servlet processor.

5.3.25 REQUEST_POINTER

```
>>-REQUEST_POINTER----------------------------------------------><
```

Returns the Mod_Rexx apache request record pointer, if running under Mod_Rexx; otherwise, the null string is returned.

Note: The `REQUEST_POINTER` may be used when writing non-portable Mod_Rexx-dependent servlets and RSPs to call Mod_Rexx-defined functions. See *Writing portable servlets* on page 47 for more information.

5.3.26 REQUEST_URI

```
>>-REQUEST_URI--------------------------------------------------><
```

Returns a string containing the unparsed request URI. This includes the query string. Compare with `URI` on page 41.

Example:

```
-- After GET /dir/servlet.rex/ path ?a= b &c=d
request~uri  -> "/dir/servlet.rex/%20path%20?a=%20b%20&c=d"
```

Note: Under Mod_Rexx the `WWWUNPARSERURI` variable has the same value.

5.3.27 SCRIPT_NAME

```
>>-SCRIPT_NAME--------------------------------------------------><
```

Returns a virtual path to the script being executed, which can be used to build self-referencing URLs.

Example:

```
-- After GET /path/servlet.rex/extra/path/info
request~script_name   -> "/path/servlet.rex"
```

Note: The value returned by this method is the value of the standard CGI environment variable `SCRIPT_NAME`. Under Mod_Rexx the `WWWSCRIPT_NAME` variable has the same value.

5.3.28 SERVER_ADDR

When running under Apache/CGI, returns the IP address of the server. Under Apache/Mod_Rexx, returns the null string.

5.3.29 SERVER_ADMIN

Under Apache, returns the value of the Apache `ServerAdmin` directive. In all other cases, the null string is returned.

Examples:

```
request~server_admin  -> "john@doe.com" /* Under apache */
request~server_admin  -> ""             /* Under IIS    */
```

Note: The value returned by this method is the value of the Apache-specific CGI environment variable `SERVER_ADMIN`. When running under Mod_Rexx, a call to `WWWSrvRecServer_admin` returns the same value.

5.3.30 SERVER_NAME

```
>>-SERVER_NAME-------------------------------------------------><
```

Returns the server's hostname, DNS alias, or IP address as it would appear in self-referencing URLs.

Note: The value returned by this method is the value of the standard CGI environment variable `SERVER_NAME`. Under Mod_Rexx the `WWWSERVER_NAME` variable has the same value.

Example:

```
request~server_name  -> "www.example.com" /* Maybe */
```

5.3.31 SERVER_PORT

```
>>-SERVER_PORT-------------------------------------------------><
```

Returns the port number to which the request was sent.

Examples:

```
request~server_port    -> 80     /* Usually   */
request~server_port    -> 443    /* For HTTPS */
```

Note: The value returned by this method is the value of the standard CGI environment variable `SERVER_PORT`. Under Mod_Rexx the `WWWSERVER_PORT` variable has the same value.

5.3.32 SERVER_PROTOCOL

```
>>-SERVER_PROTOCOL---------------------------------------------><
```

Returns the name and revision of the information protocol this request came in with, in the following format: protocol/revision.

Example:

```
request~server_protocol -> "HTTP/1.1"
```

Note: The value returned by this method is the value of the standard CGI environment variable `SERVER_PROTOCOL`. Under Mod_Rexx the `WWWSERVER_PROTOCOL` variable has the same value.

5.3.33 SERVER_SIGNATURE

Under Apache/CGI, returns the value of the Apache-specific `SERVER_SIGNATURE` CGI variable. In all other cases, the null string is returned.

```
>>-SERVER_SIGNATURE--------------------------------------------><
```

Example:

```
request~server_signature
             -> "Apache/1.3.35 Server at www.example.com Port 80"
```

5.3.34 SERVER_SOFTWARE

```
>>-SERVER_SOFTWARE---------------------------------------------><
```

Returns a string containing the name and version of the information server software answering the request (and running the gateway), in the following format: name/version

Example:

```
request~server_software -> "Apache/2.0.58 (Win32) Mod_Rexx/2.1.0"
```

Note: The value returned by this method is the value of the standard CGI environment variable `SERVER_SOFTWARE`. Under Mod_Rexx the `WWWSERVER_SOFTWARE` variable has the same value.

5.3.35 SERVLET_PROCESSOR

```
>>-SERVLET_PROCESSOR-------------------------------------------><
```

Returns the fully qualified stream name for the current RexxHttp Servlet processor, as obtained by the `PARSE SOURCE` instruction when run by the Servlet processor.

Example:

```
request~servlet_processor  -> "c:\apache\cgi-bin\RexxHttp.rex"
```

5.3.36 SYSTEM_VERSION

Returns a string identifying the RexxHttp version. This string is always of the form

"REXXHTTP/*release*.*version additional_info*"

For the current version, this is `"REXXHTTP/0.1 20061101"`.

5.3.37 UNKNOWN

```
>>-UNKNOWN(messagename,messageargs)----------------------------><
```

Returns the value of an environment variable. *Messageargs* should always be an array with no elements. *Messagename* may be of the form `HTTP_`*header* or *_variablename*. See also the description for the `[]` method on page 30.

Examples:

```
request~http_keep_alive    -> 300 /* Maybe */
request~_rspcompilers      -> 2   /* Maybe */
```

5.3.38 UNPARSEDURI

```
>>-UNPARSEDURI-------------------------------------------------><
```

This method is the same that `REQUEST_URI`. See `REQUEST_URI`.

Note: Contrast with `URI`, which follows.

5.3.39 URI

```
>>-URI---------------------------------------------------------><
```

Returns a string containing the translated uri, minus the query string, if any.

Example:

```
-- After GET /path/somefile.rex/ extrapath?a=b
request~unparseduri -> "/path/somefile.rex/%20extra%20path?a=b"
request~uri         -> "/path/somefile.rex/ extra path"
```

5.4 The Http.Response class

A Http.Response object assists a servlet in sending a response to the requester. RexxHttp automatically creates a response object for each incoming request and passes it as the second argument to the Rexx servlet. The response includes an associated Http.OutputStream object where the response contents should be written.

5.4.1 INIT

```
>>-INIT(request)-----------------------------------------------><
```

Initializes the response with the provided request value. *Request* is the Http.Request object representing the request. A Http.OutputStream is created and initialized with the request and the response, and the Content-Type header is set to `"text/plain"` (this can be changed later by the servlet if so desired).

Note: RexxHttp automatically creates a Http.Response which is passed as the second argument to every Rexx servlet.

5.4.2 []

```
>>-[header_name]-----------------------------------------------><
```

Translates to uppercase the *header_name*, which should be a string, and then returns the value of the corresponding HTTP header if it has been set. If the header has not been set, the method returns the NIL object.

Example:

```
response["Content-Type"]        --> "text/plain" /* Maybe */
```

Note: HTTP header values can be accessed using the `[]` method, or using the `response~header_name` syntax. See UNKNOWN on page 44 for more details.

5.4.3 []=

```
>>-[header_name]=header_value----------------------------------><
```

Translates to uppercase the *header_name*, which should be a string, and then sets the value of the corresponding HTTP response header to *header_value*, which should also be a string. If the header had previously been set, the old value is replaced with the new *header_value*.

Example:

```
response["Content-Type"] = "text/html"
```

Note: HTTP response header values can be set by using the `[]=` method, or by using the `response~header_name=value` syntax. See UNKNOWN on page 44.

5.4.4 ADDCOKIE

```
>>-ADDCOOKIE(cookie)-------------------------------------------><
```

Adds a copy of the Http.Cookie *cookie* to the response to be sent to the requester.

Example:

```
-- Create a new cookie
c = Http.Cookie~new("MyCookie","CookieValue")

-- Set some cookie attributes
c~max_age = 120 /* Cookie will last 2 minutes */

-- Add the cookie to the response
response~addcookie(c)
```

Note: Since a *copy* of the cookie is added to the response and not the cookie itself, further changes to the original cookie will have no impact on the response.

5.4.5 COMMIT

```
>>-COMMIT------------------------------------------------------><
```

Commits the response by sending the HTTP headers and cookies to the requester. If the response has already been committed this method has no effect.

5.4.6 COMMITTED

```
>>-COMMITTED---------------------------------------------------><
```

Returns `1` (true) if the response has been committed, and `0` (false) otherwise.

5.4.7 FLUSH

```
>>-FLUSH-------------------------------------------------------><
```

Commits the response in case it had not been commited previously, and then flushes the associated stream.

Note and example: This method can be used to immediately send part of the response contents to the requester. For example, a web application can send partial results of a long-running computation to the requester as soon as they are obtained:

```
-- Do some calculations
Say "(Some preliminary results)"

-- This sends the partial page back to the user’s browser
response~flush
```

```
-- Do some more calculations
Say "(Some further results)"

-- Send back these
response~flush

...
```

5.4.8 OUTPUT

```
>>-OUTPUT-------------------------------------------------------><
```

Returns the associated Http.OutputStream object that is used to write the contents of the response. See *The Http.Response class* on page 42 for more details.

Example:

```
-- Get the output object
out = response~output

-- Write some data to the obtained stream
out~charout(datachunk)
```

Note: RexxHttp automatically sets the destination of the standard `.output` object to be the response's output object. Under CGI, all output activity directed to the standard output stream will be directed to the response content: for example, the `SAY` instruction makes implicit use of `.output`, and therefore CGI servlets can produce their output directly as if they were normal console programs. On the other hand, Mod_Rexx traps the `SAY` instruction, and therefore the `response~output` object has to be used to attain the desired results. Of course this second method also works under CGI. See *Portable input/output* on page 47 for more details.

5.4.9 UNKNOWN

```
>>-UNKNOWN(method_name,method_args)----------------------------><
```

Allows for a syntax alternative to the `[]` and `[]=` methods.

If *method_name* does not end with "=", *method_args* should be an empty array. All occurrences of "_" in *method_name* are replaced by "-"; then the HTTP header corresponding to the resulting value is returned.

If *method_name* ends with "=", *method_args* should be an array with a single string element. *Method_name* is stripped of its trailing "=", and all occurrences of "_" are replaced by "-"; then the HTTP header corresponding to the resulting value is created or updated with the argument value.

In all other cases a syntax error is raised.

Examples:

```
response~content_type
-- Same as response["Content-Type"]

response~content_type = "text/plain"
-- Same as response["Content-Type"] = "text/plain"
```

Chapter 6

Writing portable servlets

By following some simple conventions, it is possible to write *portable servlets*. A servlet is *portable* when it runs unchanged across gateway interfaces (i.e., Mod_Rexx and CGI) and HTTP servers (i.e., Apache, IIS, etc.).

Note: Although it is also possible, by following certain provisions, to program servlets in such a manner that they are also independent of the operating system (Windows, OS/2, Linux, etc.), we will not enter into this aspect of the problem here.

Here's a short list of recommendations to follow if you want to write portable servlets:

1. Don't use command instructions addressed to `Address COMMAND`, since the Mod_Rexx processor cannot handle them.
2. Never use instructions which manipulate `STDOUT` directly (e.g., the `SAY` instruction) – always use the `response~output` stream explicitly.
3. If your servlet is handling a `POST` request, assume that the response will be flushed at the time of your first `request~arg` method invocation.
4. Use only portable request methods (a list is provided below).
5. If you need to use passed environment variables, do it in a portable way (i.e., follow the RexxHttp naming conventions).

The following sections provide more details about points 2-5 above.

6.1 Portable input/output

The input/output models of CGI and Mod_Rexx are radically different: while CGI works with `STDOUT`, and therefore implements all ooRexx-defined ways to interact with `STDOUT` (i.e., the `SAY` instruction, but also `LINEOUT`, `CHAROUT`, etc.), Mod_Rexx takes a different approach: trapping the `SAY` instruction, and effectively disabling all other methods of output to `STDOUT`.

RexxHttp, on the other hand, uses the powerful standard file redirection capabilities of ooRexx to implement a stackable, buffered output system that allows for the lazy writing of headers and cookies. Thus, RexxHttp allows, in principle, the use of all ooRexx possibilities of interacting with `STDOUT`. However, if a servlet is run under Mod_Rexx, and since the `SAY` instruction is trapped, using `SAY` directly would bypass the buffering provided by the Http.OutputStream, and possibly emit the page contents before the headers, thus causing an internal server error.

Therefore *the* `SAY` *instruction and all builtin functions that implicitly reference* `STDOUT` *cannot be used when writing servlets under Mod_Rexx.* Hence, the way to write I/O-portable Rexx servlets is to always make explicit use of the Http.OutputStream object returned by the `OUTPUT` method of the request.

6.2 Portable POST request processing

Mod_Rexx commits the response when you call `wwwGetArgs`. When running under Mod_Rexx, RexxHttp does not need to call this function to be able to parse query parameters when the request method is GET; however, a call to `wwwGetArgs` is needed when the request method is POST, and therefore RexxHttp automatically commits the response (if it was not previously committed) when a syntactically valid call to the `ARG` method is made.

To write portable servlets that process POST requests, take into account that the response may be implicitly committed when using the `ARG` method for the first time.

6.3 Portable request methods

Because of the differences between implementations, CGI and Mod_Rexx pass different sets of variables to the called Rexx program. RexxHttp passes most (not all; see below) variables to the servlet, but, of course, is not able in the general case to provide a value under Mod_Rexx for a variable which exists only under CGI, and viceversa. The same is true of the differences between the CGI variables passed by different servers (for example, Apache and IIS).

The following tables group all Http.Request methods in two categories: *portable* methods will work under all gateway interfaces and all HTTP servers; *non-portable* methods are CGI (or Apache, IIS, etc.) specific, or Mod_Rexx specific, and should only be used in the knowledge that you are writing a non-portable servlet.

The following method calls are *portable*:

Method	Mod_Rexx variable	Notes
`ARG`	N/A	
`AUTH_TYPE`	`WWWAUTH_TYPE`	
`CONTENT_LENGTH`	`WWWCONTENT_LENGTH`	
`CONTENT_TYPE`	`WWWCONTENT_TYPE`	
`COOKIE`	N/A	
`DEFAULT_TYPE`	`WWWDEFAULT_TYPE`	
`FILENAME`	`WWWFILENAME`	(1)
`GATEWAY_INTERFACE`	`WWWGATEWAY_INTERFACE`	(2)
`HTTP_xxxx`	N/A	
`METHOD`	`WWWREQUEST_METHOD`	(3)
`MOD_REXX`	(None)	(4)
`PATH_INFO`	`WWWPATH_INFO`	
`PATH_TRANSLATED`	`WWWPATH_TRANSLATED`	(1) (5)
`POST_STRING`	`WWWPOST_STRING`	(6)
`REMOTE_ADDR`	`WWWREMOTE_ADDR`	
`REMOTE_HOST`	`WWWREMOTE_HOST`	
`REMOTE_IDENT`	`WWWREMOTE_IDENT`	
`REMOTE_USER`	`WWWREMOTE_USER`	
`REQUEST_METHOD`	`WWWREQUEST_METHOD`	(7)
`REQUEST_URI`	`WWWUNPARSERURI`	
`SCRIPT_NAME`	`WWWSCRIPT_NAME`	
`SERVER_NAME`	`WWWSERVER_NAME`	
`SERVER_PORT`	`WWWSERVER_PORT`	
`SERVER_PROTOCOL`	`WWWSERVER_PROTOCOL`	
`SERVER_SOFTWARE`	`WWWSERVER_SOFTWARE`	
`SERVLET_PROCESSOR`	(None)	
`UNPARSEDURI`	`WWWUNPARSEDURI`	
`URI`	`WWWUNPARSEDURI`	

Notes: (1) Under Windows and OS/2, the path separator is `"\"` and not `"/"` as in Mod_Rexx. (2) Value is different when running under Mod_Rexx or CGI. (3) Alias for `REQUEST_METHOD`. (4) `1` under Mod_Rexx, `0` otherwise. (5) Includes the pathname and also the file name and extension. (6) Always `""` if the `REQUEST_METHOD` is not `POST`, or before the first call to `ARG`. (7) Mod_Rexx automatically handles HEAD requests without passing them to RexxHttp. The same is not true of CGI.

The following methods are *non-portable*:

Method	Notes
`REMOTE_PORT`	Apache-Specific. `""` under Mod_Rexx
`REQUEST_POINTER`	Mod_Rexx specific. `""` under CGI.
`SERVER_ADDR`	Apache-Specific. `""` under Mod_Rexx
`SERVER_ADMIN`	Apache-Specific. `""` under Mod_Rexx
`SERVER_SIGNATURE`	Apache-Specific. `""` under Mod_Rexx

The following standard Mod_Rexx variables are not accesible under RexxHttp:

Variable	Notes
`WWWFNAMETEMPLATE`	Set the appropriate variable, and use method `_FNAMETEMPLATE` instead.
`WWWHOSTNAME`	Use `SERVER_NAME` instead.
`WWWRSPCOMPILER`	See *Specifying page compilers* on page 8.

6.4 Portable variable passing

CGI environment variables are accessible by using the `VALUE` builtin function, but Apache limits their syntax to use only alpabetic characters, numbers and underscores. Mod_Rexx variables are directly accesible to the Mod_Rexx application, but unfortunately they are not global (i.e., following standard Rexx semantics, they are only available to the main program). Furthermore, there is no way under Mod_Rexx to get a list of all the passed `RexxSetVar` variables.

Hence the limitations imposed on the form of passed variables (see *Passing variables* on page 6), and the following rule, which should now be obvious: if you want to write portable servlets, use only uppercase letters, numbers and underscores for the environment variables you intend to use in your application, make sure that these variables all begin with an underscore, and always access these variables by using the `request~_`*variable_name* (or `request["_`*variable_name*`"]`) syntax.

Chapter 7

Page compiler interface

7.1 Page compilation interface

RexxHttp provides a simple interface to accomodate the use of different page compilers. When RexxHttp detects that a file must be compiled, it calls the appropriate *page_compiler* in the following way:

`Call` *page_compiler rsp_file, temporary_file, request*

Where *rsp_file* is the file to compile, *temporary_file* is the fully qualified name of a temporary unopened stream, and *request* is the Http.Request object. The page compiler may choose to compile the RSP page to a servlet in the temporary file, or may use some other mechanism and return a different file. If page compilation proceeds without errors, the page compiler executes the following instruction.

`Return 0` *compiled_servlet*

The return code is `0`, indicating that compilation proceeded without errors; the *compiled_servlet* may be identical to the passed *temporary_file*, in which case the *temporary_file* is called and subsequently erased, or different to the passed *temporary_file*, in which case the *compiled_servlet* is called and no erasing occurs.

If the page compiler encounters any kind of error which would prevent succesful execution of the *rsp_file*, it returns as follows:

`Return` *rc error_message*

The return code *rc* must be a non-zero whole number, and the *error_message* must be a descriptive string indicating the cause of the error. In this case, RexxHttp emits the return code and the *error_message* as the response and then exits abnormally.

Appendices

Appendix A

Running RexxHttp under OS/2

RexxHttp runs under OS/2 without any modification, provided that Object Rexx is active.[1] Rexx programs in OS/2 should have a file extension of `.cmd`, not `.rex`, and therefore you should change the extension of `RexxHttp.rex` to `.cmd` to be able to run RexxHttp under CGI (this change is not necessary if you are running RexxHttp under Mod_Rexx).

Contrary to Windows and Unix Object Rexx implementations, Object Rexx for OS/2 has a system-wide global `.environment` directory. RexxHttp includes a special optimization when running under OS/2: you can enhance the performance of RexxHttp by pre-loading the support classes in a separate command window; RexxHttp automatically detects that these classes are preloaded, and does not have to reload them at each RexxHttp invocation. Here is a sample code snippet:

```
.environment["HTTP.COOKIE"]        = "Http.Cookie.cls"()
.environment["HTTP.REQUEST"]       = "Http.Request.cls"()
.environment["HTTP.OUTPUTSTREAM"]  = "Http.OutputStream.cls"()
.environment["HTTP.RESPONSE"]      = "Http.Response.cls"()
.environment["HTTP.MUTABLEBUFFER"] =,
   "Http.Aux.SimpleMutableBuffer.cls"()
-- Wait forever
Parse pull .
```

The last instruction asks for a line of output and keeps the Rexx program running. This way the reference count of the support classes does not drop to zero and they are not garbage collected, thus making them available to RexxHttp.

[1] Tested on Warp Server for e-Business 4.52 and Object Rexx `OBJREXX 6.00 18 May 1999`.

Appendix B

Running RexxHttp under Microsoft IIS

Note: IIS support for RexxHttp should be considered experimental at this time.

RexxHttp detects when it is running under Microsoft IIS[1], and automatically adapts to this server. You should manually associate RexxHttp to the desired file extensions: open the administration console, right-click on the desired directory, click "Settings", press the "Configuration" button, and associate the file extensions to

ooRexx_executable RexxHttp `%s %s`

for example

```
c:\apps\ooRexx\rexx.exe c:\InetPub\test\RexxHttp.rex %s %s
```

Execution rights should be set to "Scripts and executables".

Note: Not all RexxHttp features work correctly under IIS. For example, if there is some extra path info, IIS returns a 404 Not Found error.

[1]Tested under Microsoft-IIS/5.1, which is included in Windows XP.

Appendix C

Running RSPCOMP under RexxHttp

Mod_Rexx includes a Rexx Server Pages compiler, `rspcomp.rex`. The RexxHttp package includes a modified sample version of RSPCOMP, `rhrspcmp.rex`, as well as modified versions of `rsptest1.rex`, `rsptest2.rex` and `rsptest3.rex`, the three sample programs provided with Mod_Rexx. The modifications are minimal and are clearly marked, so that you can study them to modify your own programs if you want to try RexxHttp. RSPCOMP is modified to implement the page compiler interface described in *Page compiler interface* in page 51; the sample programs are modified to follow the servlet portability guidelines of *Writing portable servlets* on page 47.

Appendix D

Running REXXTAGS under RexxHttp

RexxHttp includes a modified version of the REXXTAGS version 1.1c Rexx Server Pages compiler, `rhrxtags.rex`. This is proof-of-concept alpha version of the still unreleased version 1.2 compiler: it has been tested to work with simple REXXTAGS pages (for example, the RexxHttp test server is a REXXTAGS web), but I have not had time to thoroughly test all the REXXTAGS features; I prefer to stabilize RexxHttp first.

REXXTAGS version 1.2 will be a transitional version of REXXTAGS designed to allow easy migration to RexxHttp. It will run under ooRexx and Object Rexx only, but will not make use of the object-oriented features of Rexx beyond the obvious adaptations necessary to run under RexxHttp. The following versions will incorporate more object-oriented features.

Appendix E

Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE (”AGREEMENT”). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT’S ACCEPTANCE OF THIS AGREEMENT.

E.1 Definitions

“Contribution” means:

1. in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
2. in the case of each subsequent Contributor:
   a. changes to the Program, and
   b. additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution ‘originates’ from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor’s behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

“Contributor” means any person or entity that distributes the Program.

“Licensed Patents‘” mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

“Program” means the Contributions distributed in accordance with this Agreement.

“Recipient” means anyone who receives the Program under this Agreement, including all Contributors.

E.2 Grant of Rights

1. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

2. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

3. Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient’s responsibility to acquire that license before distributing the Program.

4. Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

E.3 Requirements

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

1. it complies with the terms and conditions of this Agreement; and

2. its license agreement:

(a) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

(b) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

(c) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

(d) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

1. it must be made available under this Agreement; and

2. a copy of this Agreement must be included with each copy of the Program. Contributors may not remove or alter any copyright notices contained within the Program. Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

E.4 Commercial Distribution

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

E.5 No Warranty

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

E.6 Disclaimer of Liability

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

E.7 General

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.