Introduction

- What's a CGI?
- How they work
- Revolution advantages
- Security
- The two ways to do Revolution CGIs
- Installing the Revolution engine
- Setting permissions

Simple CGIs

- The structure of a CGI script
- Things to keep in mind
- First CGI: "Hello World"
- Troubleshooting Tips
- Creating files on the server
- Example: Expanded "Hello World"

Working with text files

- Example: Visitor counter
- Example: Random content

Working with stacks

- Example: Using stacks with CGIs
- Setting up the files
- Use of environment variables
- Parsing URL-encoded parameters
- Putting the stack in use
- Creating HTML from within the script

Using stacks as libraries

- Using the library command
- Advantages of the library method
- Changing the Addresses example
- The LibCGI library

Appendix: Debugging text-based CGIs

- Debugging Revolution CGIs
- Quick Checklist
- Other Debugging Techniques
- Determining Environment Variables
- Other Online References

Introduction to Revolution CGIs - A Tutorial

Working with Stacks

You can access and use any data that is stored in a stack. You can also put a stack in use and access all its scripts and functionality. It is up to you whether you want to do the bulk of the work in the CGI text file, or whether you want to simply start using the stack and use its scripts from there. In general it is harder to debug text files of any complexity, and many people choose to use the second method more than the first. In this example, the work is done within the CGI text file. The next example will show how to do all the work from within the stack itself.

A brief list of things to keep in mind when using stacks with CGIs:

- Stacks must be put in use to be recognized.
- Set the defaultstack, or refer to all objects by their long names.
- Examine the environment variable `$REQUEST_METHOD` if necessary to see how form parameters are being sent. For the "get" method, read `$QUERY_STRING`. For the "post" method, read `from stdin until empty`.
- Use the `split` command and the `urlDecode` function to parse script parameters.

These items are discussed in more detail below.

Setting up the files

This example uses an HTML form to ask the user for search critera, and then the CGI script opens and searches a database stack for records that match that criteria. The matching results are returned to the user's browser.

You will need a database stack to work with. If you don't have one handy, you can download our sample five-card address stack. Place this stack in the cgi folder on the server and set its permissions to 755.

You will also need an HTML form that sends the CGI request. Create a new HTML file (or download it here) and include a form like the following. If you are running from a remote server, replace `localhost` in the "form action" line with the IP or domain address of your server.

```
<html>
<head>
        <title>Address Search</title>
</head>
<body>
<form action="http://localhost/cgi-bin/addressSearch.cgi"
method="get">
<p>
<h2>Address Database Search</h2>
Input a search word or phrase. Only whole matches are found.
<p>
<input type="text" name="terms"
value="enter search words" size="35">
<p>
<input checked name="searchScope" type=radio value="All">All Text


    
<input name="searchScope" type=radio value="byfield">By Field

<select multiple name="Fields">
        <option>Name</option>
        <option>Address</option>
        <option>City</option>
        <option>State</option>
        <option>Zip</option>
        <option>Phone</option>
</select>
<p>
<input TYPE="reset" VALUE="Reset">  <input type="submit" value="Search">
</form>
</body>
</html>
```

A brief digression: the HTML form, `post` *and* `get`

This form uses the `get` action method, but you can also use the `post` method if you prefer. The primary differences between the two are:

- the `get` method sends all its parameters as part of the browser's request to the server. The `post` method sends parameters via standard input (`stdin`).
- The `get` method is usually logged and its parameters are visible in the browser's location bar. Parameters in the `post` method are not logged and are not visible in the browser's location bar.

To read data sent with the `get` method, examine the `$QUERY_STRING` environment variable. To read data sent by the `post` method, use `read from stdin until empty`.

To ensure complete data retrieval on slow servers, it is safer to read input in a repeat loop:

```
put empty into buffer
repeat until length(buffer) = $CONTENT_LENGTH
  read from stdin until empty
  put it after buffer
end repeat
```

Regardless of how parameters are passed, they will be encoded and must be parsed. Parameters are passed as value pairs, each pair separated by an ampersand (&). Each pair consists of the name of the parameter and an equals sign, followed by the actual value, like this:

```
name=jane+doe&os=mac&color=blue
```

Use the `split` command and the `urlDecode` function for easy parsing. `Split` converts the parameters into an array, and `urlDecode` converts their format back to standard ASCII (i.e., plus signs are replaced with spaces, hex values are replaced by alpha characters, etc.)

We'll see more about URL decoding later in this example.

Back to the script

Place the HTML file containing this form into the main web folder on the server. For most ISPs, this is a folder called "www" or "public_html" or similar. In Mac OS X, this is the Documents folder inside the WebServer folder, located at `/Library/WebServer/Documents/`.

Finally, you will need the CGI script itself. Create a text file with the following script (or download it here) and install it into the cgi folder on your server. Remember to set its permissions.

```
#!revolution ui

on startup
  put $QUERY_STRING into theTerms
  put "" into buffer
  put "addresses.rev" into theStack
  if theTerms = "" then
    put "No query submitted." after buffer
  else if there is no stack theStack then
    put "Data stack cannot be found." after buffer
  else
    start using stack theStack
    set the defaultstack to theStack
    put 0 into theCt
    put theTerms into theTermsArray
    split theTermsArray by "&" and "="
    put urlDecode(theTermsArray["terms"]) into theSearchWords
    put urlDecode(theTermsArray["searchScope"]) into theScope
    unmark all cds

    if theScope = "byfield" then
      delete char 1 to offset("&Fields",theTerms) of theTerms
      replace "Fields=" with empty in theTerms
      replace "&" with comma in theTerms
      put theTerms into theFldList
      repeat for each item i in theFldList
        mark cds by finding theSearchWords in fld i
      end repeat
    else -- find in all flds
      mark cds by finding theSearchWords
    end if
    repeat with x = 1 to the number of marked cds
      put fld "name" of marked cd x & "<br>" & cr after buffer
      put fld "address" of marked cd x & "<br>" & cr after buffer
      put fld "city" of marked cd x && \
      fld "state" of marked cd x && \
      fld "zip" of marked cd x  & "<br>" & cr after buffer
      put fld "phone" of marked cd x & "<p>" & cr after buffer
      add 1 to theCt
    end repeat
    stop using stack theStack
  end if

  put "<h3>Results for " &quote& theSearchWords &quote& colon \
      & "</h3>" & theCt && "found" & "<p>" before buffer
  put "</body> </html>" after buffer

  put "Content-Type: text/html" & cr & cr
  put buffer
end startup
```

You are ready to try the CGI now. Load the HTML form page into your browser. Type "John Jones" into the search field, click the "By Field" button, and choose the "Name" option from the list. Click the Search button and you will see the results. You can also try searching for "MN" in the State field, "Broadway" in the Address field, or "Melissa" or just "Jones" in the Name field to get other results.

Some notes about the search CGI

Use of environment variables

The CGI script makes use of the environment variable `$QUERY_STRING`, whose value is set by the server when it receives the CGI request. This variable contains the parameters (that is, the search words and field options) that the user entered into the search form in their browser.

These parameters are received by the CGI in a URL-encoded form like this:

```
terms=john+jones&searchScope=byfield&Fields=name
```

Parsing URL-encoded parameters

After some basic error-checking to make sure that the search terms aren't empty and that the Addresses stack exists, the CGI script unravels these parameters by separating the various pieces into elements of an array using the `split` command, and then uses the `urlDecode` command on each element in the array in order to convert the HTML coding into regular text. The converted parameters are then placed back into script variables for use. This is generally the fastest way to parse the terms that are sent in URL-encoded form.

The only parameter that is not treated as an element in the array is the list of fields to search. In the case where more than one search field has been selected, the original URL-encoded string will list each field parameter separately, like this:

```
field=name&field=address&field=city
```

Putting this list into an array using the `split` command would cause all but one of the field names to be lost, since `split` will replace any same-named key with each new value it encounters. To accomodate this behavior, the CGI script simply deletes the first part of the parameter string, leaving only the field names behind. The word "field" and its accompanying equals sign are removed from the string, the ampersands are replaced with commas, leaving only the field names in a comma-delimited list. This is a form that the repeat loop can use later in the script.

Putting the stack in use

One of the most crucial parts of the script is the line:

```
start using stack theStack
```

The `start using` command is essential whenever you wish to use a stack with a CGI script. This command loads the required stack into memory and makes it available for use. It is in many ways the equivalent of the `go` command. Since `go` fails when the GUI is not available, `start using` is the alternative.

Note that you can also issue the `library` command instead of `start using`. These commands are equivalent.

Note also that the script sets the `defaultstack`. This is the easiest way to work with stack objects, though a script could alternately use long IDs or long names to reference stack fields and other objects.

Searching and creating HTML from within the script

The CGI searches the stack by marking those cards that contain the search terms in the requested fields, and then loops through each marked card, building HTML text out of each card's field contents. The technique is simple text chunking that combines the field contents with various HTML tags. The HTML text is stored in a variable. A text chunk containing an HTML title is inserted before the variable, and closing body and HTML tags are added after it. This is the HTML that is returned to the user's browser. Note that the "Content-type" header specifies "text/html" so that the browser will know how to display it.

The next example uses the same HTML form and Addresses stack, but moves the actual work of the CGI out of the text file and into the stack script itself.

Up to top | Introduction to Revolution CGIs - TOC | Next Page