Escapade Beta 0.2.1 Programming/Implementation Manual Version 0.8 06-07-02 @ 23:00

Escapade provides a powerful and flexible mechanism to construct dynamic web content on Linux and UNIX web servers. Escapade is similar in form and function to BASIC (or an HTML version of BASIC), except that the syntactical structure is more rigid, which makes Escapade extremely fast.

Escapade is typically used to construct dynamic web pages from data supplied in a database (MySQL is the only database that is supported at this time, but support for other databases is planned).

Escapade coexists side-by-side with HTML and is similar to HTML in form. Escapade provides for decisions and recordset looping within a page. Nesting is not fully supported (yet).

Back to Top

How Does It Work?

Escapade is installed in the server's cgi-bin directory (at this time, no support is provided for Apache plug-in capability, but this capability is planned for the near future), and is run with the server page being passed in the PAGE variable:

```
http://www.server.com/cgi-bin/esp?PAGE=do_this_page.esp
```

Pages need not end with the .esp extension, but must not contain "..". The default directory for Escapade pages is /home/httpd /cgi-bin/esp-pages.

The Escapade processor reads the page, interpreting commands, and outputs the resulting HTML to stdout (which is then passed along to the user). Any text that is not recognized as an Escapade command is ignored, and passed to the user.

While Escapade provides much of the functionality of Microsoft's Active Server Pages (ASP), it is completely different in form, which allows both to coexist. Escapade is also much simpler, making it orders of magnitude easier for the beginning web or HTML programmer to understand than PERL or other similar languages. Escapade's overhead is minimal and Escapade is much faster than PERL, owing to its stricter syntax and relative simplicity compared to PERL.

While Escapade is much simpler than PERL, it is easier and faster to implement, and consequently programmers and developers can be more productive.

Back to Top

Revisions

Ver 0.2: Added MID command.

Ver 0.3: Added PID variable, SHOWBODY, SHOWREPLY, and SHOWHEADER commands.

Added documentation for esp_run.

Ver 0.4:

Ver 0.5: Cleaned up manual a bit for first beta release, added RUNCMD, SHOWVARS commands.

Back to Top

Escapade Variable Syntax

All variables beginning with "" are interpreted by Escapade *before* any other command execution. All server-side variables are available, as are all GET and POST variables (if they are preceded by a "" sign). The built-in variables supplied by Escapade are as follows:

LONGNOW	The current time (in seconds, from January 1, 1970)
H	The current hour (one or two digits, 24 hour format)
HH	The current hour (two digits, 24 hour format)
HHH	The current hour (one or two digits, 12 hour format)
N	The current minute (one or two digits)

D The current day of the month (one or two digits)

DD The current day of the month (two digits)

YY The current year (last two digits)

YYYY The current year (four digits)

WD The current day of the week (one digit, Sunday=0)

MONTH The current month name

DAY The current day of the week name

TOD Either "morning", "afternoon", or "evening"

AMPM Either "AM" or "PM"

RECORDCOUNT Number of records returned from a statement

PID Process ID of the currently running Escapade process

DOL

The following text appearing anywhere in an Escapade script will cause the current day, time, and date to be displayed in the user’s browser:

```
Today is Thursday, April 29, 2004
```

Back to Top

Escapade Command Syntax

Escapade is composed of a handful of commands. Please note that Escapade is case-sensitive, and all commands must be in upper-case format:

<DEBUG xx> where xx is either ON or OFF

This command turns on extensive debug output. A <**DEBUG ON**> command remains in effect until terminated with a <**DEBUG OFF**> command. All debug statements are output as HTML comments. The full debug output is useful for debugging Escapade internally, although Escapade programmers will find it useful.

#

If this character is the FIRST character on any line, it is ignored and not transmitted to the browser (unlike the HTML client string <!--...-->)

<DBOPEN "database", "host", "name", "password">

This command opens the selected database on the selected host. Missing name and password parameters default to NULL, missing database and host parameters default to "mysql" and "localhost", respectively.

Note that if a <**SQL**> command is encountered before a <**DBOPEN**> statement, a <**DBOPEN**> statement with no parameters is executed automatically.

<SQL statement>
...
</SQL>

The SQL statement "statement" is executed, and Escapade loops between the following statement and </**SQL**> until all records are read. The RECORDCOUNT variable contains the number of records returned. For example, if a table called "test" contains the following:

```
+--------------+
| a1 | a2 | a3 | columns
+----+----+----+
| one| two| thr| data
| f1 | f2 | f3 | data
+--------------+

<SQL select * from test>
        The fields are , , and .
</SQL>
```

The following text would be generated:

```
The fields are one, two, and thr.
The fields are f1, f2, and f3.
```

<STOP>

All processing is stopped, and the HTML output file is closed after outputting the string "</BODY></HTML>".

The variable VAR is assigned the value of "text". Note that the quotes must exist.

```
<LET VAR="string1"X"string2">
```

The variable VAR is assigned the result of the arithmetic operation ("string1" X "string2"), where X is either +, -, /, or * (addition, subtraction, division, or multiplication). The strings are first converted to floating point format, the arithmetic operation is done, and then the result is converted back to the most precise logical format. For example:

```
<LET XXX="2"+"4"> assigns "6" to XXX

<LET XXX="3"/"2"> assigns "1.5" to XXX

<LET XXX="2.00"*"4.00"> assigns "8" to XXX
```

<FORMAT VAR "format">

The variable VAR is formatted according to the "format" string. The formats supported are "%s", "%d", "%ld", and "%f" (and modifiers). See the printf(3) man page for an explanation of these formats.

<EVAL XXX "YYY">

Assigns the value of "YYY" to "XXX". This is useful when you have a variable name that is dependent on the value of another variable. For example, suppose you had the following variables set:

```
<LET A1="12345">

<LET A2="23456">

<LET A3="67890">

<LET ID="2">
```

If you wanted to determine the value of either A1, A2, or A3, but the variable name was dependent on the value of ID, you could assign OUT like:

```
<EVAL OUT "A">
```

Escapade would evaluate "A" to "A2", then fetch the value of the variable "A2" and place that value into OUT.

<MID VAR "string" "start" "length">

Assigns to VAR the substring of "string" starting at "start" (origin 1) for "length" characters. If "length" is omitted, the rest of the string is assigned. For example:

```
<MID ABC "gobbledegook", "5", "3">

<MID DEF "gobbledegook", "7">
```

will assign the string "led" to the variable ABC, and the string "degook" to the variable DEF.

<MD5 VAR "string">

Calculate an MD5 cryptographic checksum of string "string" and assign the value to VAR.

<SHOWFILE file>

Converts the named "file" to HTML format and sends it to the browser.

<SHOWHTML file>

Sends "file" to the browser. No conversion is done.

<SHOWMAIL file>

Converts "file" to HTML format and sends it to the browser. "http://" and "mailto:" strings are automatically converted to hyperlinks. The file text up to the first blank line is discarded.

<SHOWEMAIL file>

Same as <SHOWMAIL>, but automatically decrypts "file" using PGP. PGPPASS and PGPPATH must be set to appropriate values (see the PGP Command Line documentation for further information). PGP must reside in /usr/local/bin/pgp (this can also be a link to somewhere else).

<SHOWBODY file>

Strips all lines up to the first blank line from "file", then sends the rest of the file to the browser. No conversion is done.

<SHOWREPLY file>

Like SHOWBODY, but prepends "> " to the front of each line.

<SHOWHEADER file>

Sends all lines up to the first blank line to the browser. No conversion is done.

Conditionally assigns "string3" to VAR. String comparisons are done, unless both "string1" and "string2" are numbers, in which case numeric comparisons are done (this only applies to "LT" and "GT" comparisons, however). "AA" can be one of the following:

LT	less than
GT	greater than
EQ	equal
NE	not equal
LI	like

If LI is specified, the assignment will be done if "string2" is not null and "string2" is found within "string1".

<IF "string1" AA "string2">
...
</IF>

Similar to the preceding <**IF**> directive, except that all following statements are executed if the condition is met, until an </IF> is encountered.

<INCLUDE file>

Includes "file" as if the text were included in the original file.

Includes cannot be nested.

<RUNCMD "command">

Run "command". Stderr output is redirected to stdout. This command may not work if an INCLUDE directive is encountered prior to the RUNCMD directive.

Back to Top

Supplementary Commands

Escapade comes with a supplementary command called "esp_run". Esp_run will automatically set in the environment strings passed with a GET or PUT so that a called script may have access to them. The script or command to be run is passed in the COMMAND variable. Example:

```
<FORM ... ACTION="/cgi-bin/esp_run">
<INPUT TYPE="hidden" NAME="COMMAND" VALUE="./sendmail">
<INPUT TYPE="hidden" NAME="ABC" VALUE="xyz">
```

This will run the "sendmail" script with all variables set in the environment (the variable "ABC" will be set to the value "xyz").

Back to Top

Tutorial

Escapade is easy to learn - if you know HTML, Escapade can be mastered in a few hours.

Here is a complete web page that will display the time and date to the user. Note that this page, named "sample1.esp", is called thusly:

```
Click <A HREF="/cgi-bin/esp?PAGE=sample1.esp">here</A> to display the time and date.
```

The actual page, "sample1.esp":

```
<HTML>
<HEAD>
<TITLE>
Sample Escapade Web Page
</TITLE>
</HEAD>
<BODY>
The current time is 11:31 and today's date is 04/29/04.
</BODY>
</HTML>
```

Note that the page is composed almost entirely of pure HTML, which is passed along to the browser. Escapade detects the variables for time and date, and fills in the values before passing the line to the browser.

How about a slightly more difficult example? Let's display the contents of a table in a MySQL database called "table1":

"table1" is composed of three fields, f1, f2, and f3. If the data in the table1 looks like:

```
+--------------------+
|  f1  |  f2  |  f3  |
+------+------+------+
```

The entire web page to display the table would look like:

```
<HTML>

<HEAD>

<TITLE>table1 output</TITLE>

</HEAD>

<BODY>

<!-- open the database with the defaults -->

<DBOPEN>

<!-- do the select -->

<SQL select * from table1>

<!-- loop through all the records -->

<!-- note that the variables are replaced with the actual field values and the user’s browser sees
none of this code -->

f1 = , f2 = , f3 = <BR>

</SQL>
```

What if we only want to see the data for Bill? Well, there are two ways to do this: (1) modify the SELECT statement:

```
<SQL select * from table1 WHERE f1='bill'>
```

or we could check while looping through the records. This is less efficient, but is a good illustration of how to make decisions on a per-record basis:

```
<SQL select * from table1>

<IF "" EQ "bill">

f1 = , f2 = , f3 = <BR>

</IF>

</SQL>
```

Back to Manual Home Top