spiffy-utils egg

Utility library for the spiffy webserver
Extension for Chicken Scheme
Version 0.4

Peter Bex

Table of Contents

1 About this egg

1.1 Version history

`0.4` Add `link-to` function for constructing URL with GET parameters. Add default and string-conversion optional arguments to variable-getters. Remove `assocval` as it is equivalent to `alist-ref`.

`0.3` Removed runtime-dependency on `match-support` unit [Felix].

`0.2` Fixed cookie quoting bug [Thanks to Michele Simionato].

`0.1` Initial version.

1.2 Requirements

This egg requires the following extensions:

`spiffy`, `url`

1.3 Usage

Load this egg like so:

```
(require-extension spiffy-utils)
```

2 Documentation

Utility library for spiffy to make advanced webprogramming more comfortable. Features include:

- RFC 2109 cookies.
- Easy accessor functions for GET-, POST- and cookie variables.
- Easy construction of URLs with GET-arguments.

`link-to` [procedure]

```
(link-to url variable-alist)
```

Construct an URL string that links to `url` and has an association list of variable/value pairs. These pairs can be extracted from the request arguments using `get-var`. Both the name and the value objects of the alist are converted to strings using `->string`.

`get-var` [procedure]

```
(get-var varname [string->type] [default])
```

Returns the value for the named variable from the GET argument list, or `default` if it could not be found. (`#f` if not specified). The returned value is a string which can optionally be converted to the desired type by supplying a `string->type` argument.

`post-var` [procedure]

```
(post-var varname [string->type] [default])
```

Same as `get-var`, but for http POST requestvariables instead.

`cookie-var` [procedure]

```
(cookie-var varname [string->type] [default])
```

Same as `get-var`, but for cookie variables instead. Cookies automatically get sent by the user agent on each request, both GET and POST, so they're always available.

`string->bool` [procedure]

```
(string->bool b)
```

Convenience function to convert the `string` value `b` to a `<boolean>`value. (useful in combination with `get-var`, `post-var` and `cookie-var`.

`string->boolean` [procedure]

```
(string->boolean b)
```

An alias for `string->bool`

`(write-cookie name value [comment] [max-age] [domain] [path] [secure])` [procedure]

Instruct Spiffy to write a cookie header in the current reply. Cookies are `name/value` pairs of strings (but any value is ok; `->string` is automatically called on the two values). A cookie has an optional comment string for the user if he has a client which allows him to view the cookies. Cookies will expire when the session ends (usually when the browser closes) unless the `max-age` is supplied, which is an integer argument declaring the maximum time the cookie can be stored on the client side, in seconds. The domain string specifies for which domain the cookie is. The cookie will only

be returned on subsequent requests to that domain. By default, the domain is the domain from which the cookie originates. The path string specifies the topmost path from which this cookie is active. If there are multiple cookies with the same name, the value from the cookie with the least generic ("deepest") path will be passed to the server. Finally, the boolean secure can be set to `#t` if the cookie should be treated as confidential information. If the user wishes to set one parameter, but not any other optional parameters, the value `#f` can be used. These parameters will be ignored.

`(delete-cookie name [domain] [path]` [procedure]

Instruct Spiffy to write a cookie header constructed in such a way that the cookie with the supplied `name` (also converted to a string with `->string`) will be deleted. The two optional parameters behave as with `write-cookie`. (See above.)

`exec` [procedure]

```
(exec cmd [disp])
```

Execute the commandline indicated by the `cmd` string and give its standard output as a string result. The optional `disp` argument will be used to display each line, so one can add HTML tags to every line of output if so desired. This allows one to write simple one-liners like:`Uptime: <? (exec "/usr/bin/uptime ") ?>`**Warning:** this function passes the commandline right to the shell, so never build it from user input.

3 Examples

```
;; A simple webpage which allows you to play with three different
;; types of variables, which are submitted to the page itself via HTTP GET.
(use spiffy-utils sxml-transforms srfi-1 regex)
(define vars `((integer . ,(get-var 'integer string->number 0))
               (boolean . ,(get-var 'boolean string->bool))
               (string . ,(get-var 'string identity ""))))

(define (addfoo var)
  (string-append var "foo "))

(define (delfoo var)
  (string-substitute "foo " "" var))

(define (alist-replace var func)
  (alist-cons var (func (alist-ref var vars))
              (alist-delete var vars)))

(define my-rules
  `((url *macro* . ,(lambda (url target text)
                      `(a (@ href ,target) ,text)))))

(define document
 `((html
    (head
     (title "GET-var and typing showcase"))
    (body
     (dl
      (dt "Integer:")
      (dd (url ,(link-to "test.ssp" (alist-replace 'integer sub1)) "sub1")
          " " ,(alist-ref 'integer vars) " "
          (url ,(link-to "test.ssp" (alist-replace 'integer add1)) "add1"))
      (dt "Boolean:")
      (dd ,(->string (alist-ref 'boolean vars)) " "
          (url ,(link-to "test.ssp" (alist-replace 'boolean not)) "Toggle"))
      (dt "String:")
      (dd (url ,(link-to "test.ssp" (alist-replace 'string delfoo)) "Delete foo")
          " " ,(alist-ref 'string vars)
          (url ,(link-to "test.ssp" (alist-replace 'string addfoo)) "Add foo")))))))

(SRV:send-reply (pre-post-order document (append my-rules universal-conversion-rules)))
```

4 License

Index