ajax egg

A basic interface to AJAX, using the Prototype javascript library (included).
Extension for Chicken Scheme
Version 1.6

felix

Table of Contents

1 About this egg

1.1 Version history

`1.6` Added key parameter to `ajax`

`1.5` Removed quoting from `remote-action` argument parameter

`1.4` Updated prototype.js to version 1.4.0

`1.3` Fixed bug in handling of 'parameters' keyword arg

`1.2` Several enhancements by Daishi Kato

`1.1` Bugfix in `remote-button` by Daishi Kato; `current-request` and `current-urlencoded-arguments` are now also available in callbacks

`1.0` Initial release

1.2 Requirements

This egg requires the following extensions:

`md5`, `spiffy`, `url`

1.3 Usage

Load this egg like so:

```
(require-extension ajax)
```

2 Documentation

This extension provides a set of procedures to use asynchronous callbacks from a browser to a Chicken-powered web-server (for example Spiffy). This is done using Prototype's wrappers around the `xmlHttpRequest` API.

`ajax-output-format` [parameter]

```
(ajax-output-format [SYMBOL])
```

Specifies how HTML and script-fragments generated by the procedures of this package should be handled. `SYMBOL` should be either `shtml` or `html`, where the former requests output as SXML expressions and the latter as text containing HTML. `html` is the default.

`ajax-callback-registration` [parameter]

```
(ajax-callback-registration [PROCEDURE])
```

Holds a procedure that will be called for every registered `xmlHttpRequest` callback (see below).

`ajax` [procedure]

```
(ajax [KEY [SCRIPTFILENAME]])
```

Initializes callback handling for this page/response and returns a HTML or SHTML fragment containing a script element linking to the given source filename, which defaults to `prototype.js`. The Prototype library is included in the distribution and will be installed in the extension repository.

The optional argument KEY is used to make the generated callback-URLs for this page more unique and may be an arbitrary string.

This procedure resets the callback-handling for this response and must be called before using any of the `remote-...` procedures described below.

`remote-url` [procedure]

```
(remote-url THUNK [ARGUMENTS])
```

Registers a HTTP callback handler for a generated URL that invokes THUNK, a procedure with no arguments. If ARGUMENTS is given, then it should be a string or an association-list mapping URL-encoded argument-names to values. In both cases the arguments will be encoded in the url, or the body of the request, depending on the request method.

This procedure returns an URL as a string.

`remote-action` [procedure]

```
(remote-action THUNK #!key arguments update insert-before insert-after insert-top
```

Returns a string containing Javascript code to invoke the zero-argument procedure THUNK via an `xmlHttpRequest` callback. Depending on the keyword arguments the result returned from THUNK is treated as HTML or SHTML and describes a HTML fragment that should be used to modify the current document.

The meaning of the various keyword arguments is as follows.

`arguments`	A string or an association list of URL-encoded variable/value pairs (see `remote-url` for more information)
`update`	the id of a document node that should be replaced with the SXML of this reply
`insert-before`	Document-id of element before which the reply will be inserted
`insert-after`	Document-id of element before which the reply will be inserted
`insert-top`	Document-id of element in which the reply will be inserted as the topmost sub-element
`insert-bottom`	Document-id of element in which the reply will be inserted as the bottommost sub-element
`method`	HTTP request method used
`success`	Javascript code to be evaluated on successful reply
`failure`	Javascript code to be evaluated on non-succesful reply
`name`	The name of a Javascript variable which should be assigned to the created Prototype placeholder object
`frequency`	The interval in seconds after which the callback should be invoked
`eval-scripts`	If true, evaluate JavaScript code received in the response

Note that the `update`, `insert-...`, `success` and `failure` arguments are mutually exclusive. If the `frequency` argument is given, a periodical timer will be started (you can use the `name` argument to obtain a reference to the timer and call the `start` and `stop` Javascript methods to control it).

`remote-link` [procedure]

```
(remote-link TEXT THUNK KEYWORD-ARGUMENTS ...)
```

Returns HTML or SHTML for a link (`<a>`) that will invoke THUNK when clicked. The KEYWORD-ARGUMENTS are the same as for `remote-action`.

`remote-button` [procedure]

```
(remote-button TEXT THUNK KEYWORD-ARGUMENTS ...)
```

Returns HTML or SHTML for a button (`<button>`).

`remote-timer` [procedure]

```
(remote-timer FREQUENCY THUNK KEYWORD-ARGUMENTS ...)
```

Returns HTML or SHTML for a Javascript element that starts a periodical timer triggered every FREQUENCY seconds.

`no-cache` [procedure]

```
(no-cache)
```

Disables caching of the current page by setting various HTTP response headers. Handy for debugging.

3 Examples

Here we use the `ajax` extension in an ssp page. To try it out you can invoke `csi` like this (assuming `prototype.js` and `test.ssp` are in the current directory):

```
$ csi -q
#;1> (use spiffy)
#;2> (start-server port: 8080 root: "." debug: #t)
```

```
<!-- test.ssp -->
<html>
<head>
<?scheme
(use ajax)
(no-cache)
(display (ajax))
(define words '(one two three))
?>
</head>
<body>
 The current time is: <span id="time"><b>...</b></span>
 <p>
 <ul id="list"></ul>
 <p>
 <?
 (remote-timer
   2
   (lambda () (printf "<b>~a</b>" (seconds->string (current-seconds))))
   update: "time")
 ?>
 <p>
 <?
 (remote-button
   "Click me"
   (lambda () (printf "<li>~a</li>" (list-ref words (random (length words)))))
   insert-bottom: "list")
 ?>
</body>
</html>
```

4 License

Index