CLAW User Manual

CLAW version 0.1
genuary 2008

Table of Contents

1 Introduction 1
1.1 What is CLAW 1
1.1.1 The request cycle 1

2 The server 3
2.1 Understanding the clawserver 3
2.1.1 CLAWSERVER instance initialization 3
2.1.2 CLAWSERVER class methods 4
2.2 Starting the server 7
2.2.1 Making CLAW work on http protocol 7
2.2.2 Making CLAW work on both http and https protocols 8
2.2.3 Making all applications to work under a common path 9

3 Lisplets 10
3.1 Registering a lisplet into the server, crating a web application . . 10
3.2 Adding resources into a LISPLET 11
3.2.1 Adding files and folders to a LISPLET 11
3.2.2 Adding functions to a LISPLET 11
3.2.3 Adding pages to a LISPLET 12

4 Web application pages 13
4.1 Writing your first CLAW page 13
4.1.1 The special tag attribute: :ID and :STATIC-ID 13

5 Getting started with CLAW 17

6 Internationalization of our application 18

7 CLAW forms and form components 19

8 Input validation and field translations 20

9 Creating a web application by writing reusable components 21

10 Writing advanced components 22

11 Access validation and authorization 23

12 Advanced techniques 24

Function index 25

1 Introduction

CLAW is a comprehensive web application for Common Lisp programming language.

1.1 What is CLAW

CLAW is a comprehensive web application framework for the Common Lisp programming language. CLAW is based on components, highly reusable building blocks the make easy and fast the creation of a web application. By using and creating new components, the developer can create robust and consistent web application with the minimal effort.

Each component may inject into a page its own set of stylesheet and javascript files, and may come with its own class or instance javascript directives (a class directive is inserted only once into the page, while this is not true for an instance script). This leads to the creation of very sophisticated components with a very little effort.

CLAW comes with its own authentication system that lets you create both basic and form based authentication systems.

CLAW has the capability to force the page rendering through the HTTPS protocol of pages managing sensible data, using simple directives.

CLAW comes with its own extensible localization and validation system.

The main aim of CLAW is *'divide et impera'*, that means that dividing problems into small problems let programmers work on different part of an application, creating ad hoc components for both generic and specific tasks.

CLAW can easily handle all the request cycle,letting you to concentrate only in application business side problems, letting CLAW automatically manage all the mechanism of the web layer, such as form submission and user interactions.

CLAW comes integrated with the dojotoolkit, giving you the possibility to easily and quickly create full WEB 2.0 eye candy application, with powerful and very user friendly UI.

1.1.1 The request cycle

When a user asks for a page the request is sent to the `CLAWSERVER` that dispatches the request to the registered lisplets.

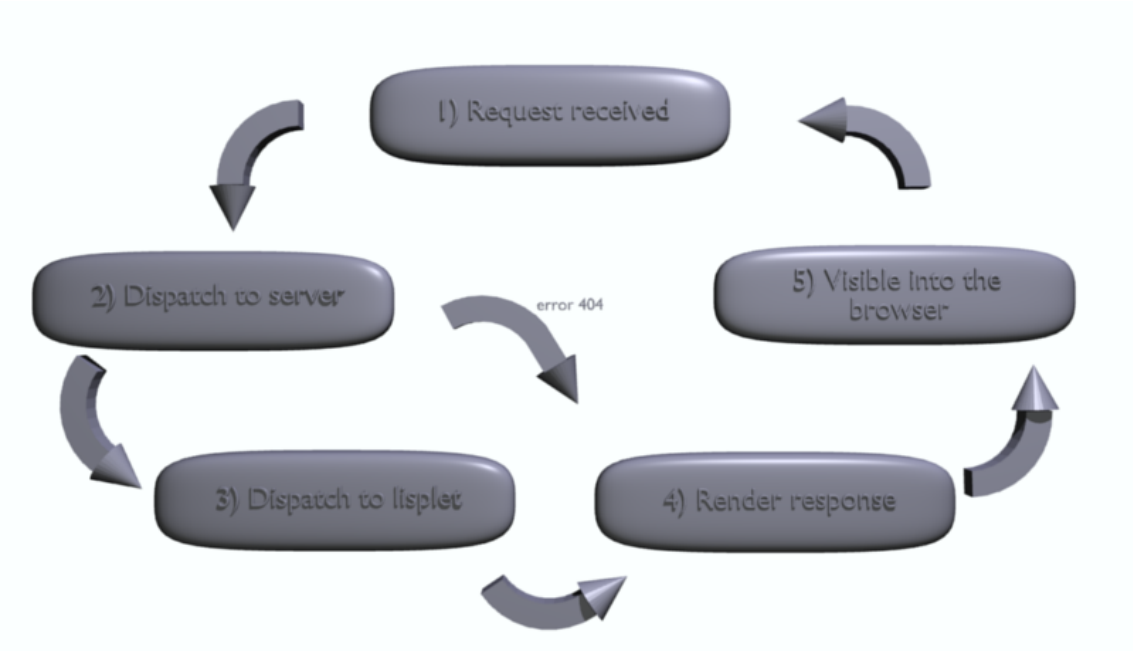
Lisplets are web resource containers that hold web pages and other resource files, such as javascript, image, css, etc. files, or even functions, under a common path.

When a matching lisplet is then found, it dispatches the request to a registered resource that can be a page or a file or even a function.

If the request is sent for a file, this is then sent back to the browser if found.

If the request is sent for a page, usually mapped to a html URL, the dispatcher calls the page rendering function to display the page as an html resource.

If no resource is found a 404 message page, is sent to the user as feedback.



2 The server

CLAW wraps the Hunchentoot (see: unchentoot), a wonderful as powerful web server written in Common Lisp, into the `CLAWSERVER` class.

As an Hunchentoot wrapper `CLAWSERVER` "provides facilities like automatic session handling (with and without cookies), logging (to Apache's log files or to a file in the file system), customizable error handling, and easy access to GET and POST parameters sent by the client."

2.1 Understanding the clawserver

`CLAWSERVER` is not only a Hunchentoot wrapper, it is also the common place where you put your web applications built with CLAW into lisplet that you can see as application resource containers and request dispatchers.

2.1.1 `CLAWSERVER` instance initialization

When you want to instantiate a `CLAWSERVER` class, remember that it accepts the following initialization arguments:

- *port* The port the server will be listening on, default is 80
- *sslport* The SSL port the server will be listening on, default is 443 if the server has a certificate file defined.
- *address* A string denoting an IP address, if provided then the server only receives connections for that address. If address is NIL, then the server will receive connections to all IP addresses on the machine (default).
- *name* Should be a symbol which can be used to name the server. This name can utilized when defining easy handlers. The default name is an uninterned symbol as returned by GENSYM
- *sslname* Should be a symbol which can be used to name the server running in SSL mode when a certificate file is provided. This name can utilized when defining easy handlers. The default name is an uninterned symbol as returned by GENSYM
- *mod-lisp-p* If true (the default is NIL), the server will act as a back-end for mod_lisp, otherwise it will be a stand-alone web server.
- *use-apache-log-p* If true (which is the default), log messages will be written to the Apache log file - this parameter has no effect if *mod-lisp-p* is NIL.
- *input-chunking-p* If true (which is the default), the server will accept request bodies without a Content-Length header if the client uses chunked transfer encoding.
- *read-timeout* Is the read timeout (in seconds) for the socket stream used by the server. The default value is HUNCHENTOOT:*DEFAULT-READ-TIMEOUT* (20 seconds)
- *write-timeout* Is the write timeout (in seconds) for the socket stream used by the server. The default value is HUNCHENTOOT:*DEFAULT-WRITE-TIMEOUT* (20 seconds)
- *setuid* On Unix systems, changes the UID of the process directly after the server has been started.
- *setgid* On Unix systems, changes the GID of the process directly after the server has been started.

– *ssl-certificate-file* If you want your server to use SSL, you must provide the pathname designator(s) for the certificate file (must be in PEM format).
– *ssl-privatekey-file* the pathname designator(s) for the private key file (must be in PEM format).
– *ssl-privatekey-password* If private key file needs a password set this parameter to the required password

2.1.2 `CLAWSERVER` class methods

`clawserver-port obj`
`(setf clawserver-port) val obj`

- *obj* The `CLAWSERVER` instance
- *val* The numeric value of listening port to assign

Returns and sets the port on which the server is listening to (default 80). If the server is started and you try to change the listening value an error will be signaled

`clawserver-sslport obj`
`(setf clawserver-sslport) val obj`

- *obj* The `CLAWSERVER` instance
- *val* The numeric value of listening port to assign for SSL connections

Returns and sets the port on which the server is listening to in SSL mode if a certificate file is provided (default 443). If the server is started and you try to change the listening value an error will be signaled

`clawserver-address obj`
`(setf clawserver-address) val obj`

- *obj* The `CLAWSERVER` instance
- *val* The string value denoting the IP address

Returns and sets the IP address where the server is bound to (default `NIL` ⇒ any). If the server is started and you try to change the listening value an error will be signaled

`clawserver-name obj`
`(setf clawserver-name) val obj`

- *obj* The `CLAWSERVER` instance
- *val* The symbol value denoting the server name

Should be a symbol which can be used to name the server. This name can utilized when defining easy handlers. The default name is an uninterned symbol as returned by GENSYM

`clawserver-sslname obj`
`(setf clawserver-sslname) val obj`

- *obj* The `CLAWSERVER` instance

- *val* The symbol value denoting the server name running in SSL mode

Should be a symbol which can be used to name the server running in SSL mode, when a certificate file is provided. This name can utilized when defining easy handlers. The default name is an uninterned symbol as returned by GENSYM

`clawserver-mod-lisp-p obj`
`(setf clawserver-mod-lisp-p) val obj`

- *obj* The `CLAWSERVER` instance
- *val* The boolean value denoting the use of mod_lisp.

Returns and sets the server startup modality . If true (the default is `NIL`), the server will act as a back-end for mod_lisp, otherwise it will be a stand-alone web server. If the server is started and you try to change the listening value an error will be signaled

`clawserver-use-apache-log-p obj`
`(setf clawserver-use-apache-log-p) val obj`

- *obj* The `CLAWSERVER` instance
- *val* The boolean value denoting the use of Apache log.

Returns and sets where the server should log messages. This parameter has no effects if clawserver-mod-lisp-p is set to `NIL`. (default `T` if `mod_lisp` is activated. If the server is started and you try to change the listening value an error will be signaled

`clawserver-input-chunking-p obj`
`(setf clawserver-input-chunking-p) val obj`

- *obj* The `CLAWSERVER` instance
- *val* The boolean value denoting the ability to accept request bodies without a Content-Length header.

Returns and sets the ability to accept request bodies without a Content-Length header (default is `T`) If the server is started and you try to change the listening value an error will be signaled

`clawserver-read-timeout obj`
`(setf clawserver-read-timeout) val obj`

- *obj* The `CLAWSERVER` instance
- *val* The integer value denoting the server read timeout.

Returns and sets the server read timeout in seconds (default is `T`) (default to HUNCHENTOOT:*DEFAULT-READ-TIMEOUT* [20 seconds]). If the server is started and you try to change the listening value an error will be signaled

`clawserver-write-timeout obj`
`(setf clawserver-write-timeout) val obj`

- *obj* The `CLAWSERVER` instance

- *val* The integer value denoting the server write timeout.

Returns and sets the server write timeout in seconds (default is `T`) (default to HUNCHENTOOT:*DEFAULT-WRITE-TIMEOUT* [20 seconds]). If the server is started and you try to change the listening value an error will be signaled

```
clawserver-setuid obj
(setf clawserver-setuid) val obj
```

- *obj* The `CLAWSERVER` instance
- *val* The string or integer value of the UID with which the server instance will run.

Returns and sets the server instance UID (user id). If the server is started and you try to change the listening value an error will be signaled

```
clawserver-setgid obj
(setf clawserver-setgid) val obj
```

- *obj* The `CLAWSERVER` instance
- *val* The string or integer value of the GID with which the server instance will run.

Returns and sets the server instance GID (group id). If the server is started and you try to change the listening value an error will be signaled

```
clawserver-ssl-certificate-file obj
(setf clawserver-ssl-certificate-file) val obj
```

- *obj* The `CLAWSERVER` instance
- *val* Pathname designator(s) for the certificate file

Returns and sets the pathname designator(s) for the certificate file if the `CLAWSERVER` is SSL enabled If the server is started and you try to change the listening value an error will be signaled

```
clawserver-ssl-privatekey-file obj
(setf clawserver-ssl-privatekey-file) val obj
```

- *obj* The `CLAWSERVER` instance
- *val* Pathname designator(s) for the private key file

Returns and sets the pathname designator(s) for the private key file if the `CLAWSERVER` is SSL enabled If the server is started and you try to change the listening value an error will be signaled

```
clawserver-ssl-privatekey-password obj
(setf clawserver-ssl-privatekey-password) val obj
```

- *obj* The `CLAWSERVER` instance
- *val* Password for the private key file

Returns and sets the password for the private key file if the CLAWSERVER is SSL enabled If the server is started and you try to change the listening value an error will be signaled

`clawserver-start obj`

- *obj* The CLAWSERVER instance

Make the CLAWSERVER begin to dispatch requests

`clawserver-stop obj`

- *obj* The CLAWSERVER instance

Make the CLAWSERVER stop.

`clawserver-register-lisplet clawserver lisplet-obj`

- *obj* The CLAWSERVER instance
- *lisplet-obj* A LISPLET class instance

Registers a LISPLET, that is an 'application container' for request dispatching.

`clawserver-unregister-lisplet clawserver lisplet-obj`

- *obj* The CLAWSERVER instance
- *lisplet-obj* A LISPLET class instance

Unregisters a LISPLET, that is an 'application container', an so all it's resources, from the CLAWSERVER instance.

2.2 Starting the server

Starting CLAW is very easy and requires a minimal effort. CLAW supports both http and https protocols, thought enabling SSL connection for CLAW requires a little more work then having it responding only to http calls.

2.2.1 Making CLAW work on http protocol

To simply start CLAW server, without enabling SSL requests handling, you just need few steps:

```
(defparameter *clawserver* (make-instance 'clawserver))
(clawserver-start *clawserver*)
```

This will start the web server on port 80 that is the default.

Of course you can create a parametrized version of CLAWSERVER instance for example specifying the listening port as the following:

```
(defparameter *clawserver* (make-instance 'clawserver :port 4242))
(clawserver-start *clawserver*)
```

2.2.2 Making CLAW work on both http and https protocols

To enable CLAW to https firt you need a certificate file. A quick way to get one on a Linux system is to use openssl to generate the a certificate PEM file, the following example explains how to do.

Firstly you'll generate the private key file:

```
#> openssl genrsa -out privkey.pem 2048

Generating RSA private key, 2048 bit long modulus
..........................+++
.............................................+++
e is 65537 (0x10001)

#>
```

Then the certificate file:

```
#> openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IT
State or Province Name (full name) [Some-State]: bla-bla
Locality Name (eg, city) []: bla-bla
Organization Name (eg, company) [Internet Widgits Pty Ltd]: mycompany
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:www.mycompany.com
Email Address []:admin@mycompany.com

#>
```

Now you can start `CLAWSERVER` in both http and https mode:

```
(defparameter *clawserver* (make-instance 'clawserver :port 4242
              :sslport 4443
              :ssl-certificate-file #P"/path/to/certificate/cacert.pem"
              :ssl-privatekey-file #P"/path/to/certificate/privkey.pem")))
(clawserver-start *clawserver*)
```

CLAW is now up and you can browse it with your browser using address http://www.yourcompany.com:4242 and http://www.yourcompany.com:4443. Of course you will have only a 404 response page!

2.2.3 Making all applications to work under a common path

You have the possibility to define a common path to mapp all CLAW applications registered into the server, defining the global variable `*CLAWSERVER-BASE-PATH*`. This way, if you have two applcations mapped for example to "/applicationA" and "/applicationB", setting that variable to the common path "/yourcompany" with the instruction

```
(setf *clawserver-base-path* "/yourcompany")
```

you will have the two applications now mapped to "/yourcompany/applicationA" and "/yourcompany/applicationB".

3 Lisplets

Lisplets are CLOS objects that extend the functionalities of CLAWSERVER, dispatching requests that come from this last one.

Lisplets are so, the place where you put your web applications developed with CLAW.

Lisplets return to the requesting user, pages, functions and resources mapped into them.

Each Lisplet contains its own dispatch table and realm so that applications are not mixed together.

3.1 Registering a lisplet into the server, crating a web application

To create a web application you have to instantiate a LISPLET and then register it into the server.

```
(defvar *clawserver* (make-instance 'clawserver :port 4242))

(defvar *test-lisplet* (make-instance 'lisplet :base-path "/test"))
(clawserver-register-lisplet *clawserver* *test-lisplet*)

;;; you can now start the server
;;; with:
;;; (clawserver-start *clawserver*)
;;; and
;;; (clawserver-stop *clawserver*)
```

At this point you have defined a web application registered to the URL "http://localhost:4242/test" that CLAW will be able to serve.

All sessions and the authentication and authorization logic will be under the default realm "claw", so if you register another lisplet into the server with the instruction:

```
(defvar *test-lisplet2* (make-instance 'lisplet :base-path "/test2"))
(clawserver-register-lisplet *clawserver* *test-lisplet2*)
```

any user session will be shared among *test-lisplet* and *test-lisplet2* and if a user is logged into "/test" application, he will be logged into "/test2" application too.

To avoid this behaviour, you need to define a different realm for each of the two lisplet as the following example does:

```
(defvar *clawserver* (make-instance 'clawserver :port 4242))

(defvar *test-lisplet* (make-instance 'lisplet :realm "test"
                                      :base-path "/test"))
(clawserver-register-lisplet *clawserver* *test-lisplet*)

(defvar *test-lisplet2* (make-instance 'lisplet :realm "test2"
                                       :base-path "/test2"))
(clawserver-register-lisplet *clawserver* *test-lisplet2*)
```

The two lisplets will now have different realms, so a user session in `*test-lisplet*` will be different from the one in `*test-lisplet2*`. So for the authentication and authorization module. The same is for a user logged into the first application, he will not be automatically logged into the other now.

3.2 Adding resources into a LISPLET

Lisplets alone don't do anything more then providing some error pages when something goes wrong. To make a LISPLET a web application, you have to fill it with some application resource, and this may be done in several ways.

3.2.1 Adding files and folders to a LISPLET

Suppose now you want to provide, thought your web application, a file present on your hard disk, for example: "/opt/webresources/images/matrix.jpg".

This is made very simple with the following instructions

```
(lisplet-register-resource-location *test-lisplet*
                                    #P"/opt/webresources/images/matrix.jpg"
                                    "images/matrix.jpg" "image/jpeg")
```

The jpeg file will now be available when accessing "http://localhost:4242/test/images/matrix.jpg". The last argument specifies the mime-type, but it's optional.

If you want to register an entire folder, the process is very similar

```
(lisplet-register-resource-location *test-lisplet*
                                    #P"/opt/webresources/images/"
                                    "images2/")
```

Now you'll be able to access the same resource following the URL "http://localhost:4242/test/images2/matrix.jpg", easy, isn't it?

3.2.2 Adding functions to a LISPLET

Registering a function gives you more flexibility then registering a static resource as a file or a directory but the complexity relies into the function that you want to register.

For example, if you want to provide the same "matrix.jpg" file through a function, you'll have to do something of the kind:

```
(lisplet-register-function-location *test-lisplet*
    #'(lambda ()
(let ((path #P"/opt/webresources/images/matrix.jpg"))
  (setf (content-type) (mime-type path))
  (with-open-file (in path :element-type 'flex:octet)
    (let ((image-data (make-array (file-length in)
  :element-type 'flex:octet)))
      (read-sequence image-data in)
      image-data))))
    "images/matrix2.jpg" )
```

Now the image will be available at the URL "http://localhost:4242/test/images/matrix2.jpg".

The method `lisplet-register-function-location` accepts two optional keys:

– `:WELCOME-PAGE-P` that will redirect you to the registered location when you'll access your application with the URL "http://localhost:4242/test"

– `:LOGIN-PAGE-P` that will redirect an unregistered user to the resource when he tries to access a protected resource to perform the login with a form based authentication.

3.2.3 Adding pages to a LISPLET

Pages are one of the key objects of CLAW, since they are sophisticated collectors of web components. Pages are described in the next chapter, meanwhile to register a page that is a `CLOS` object, the procedure is very similar to when you register a function.

```
(defclass empty-page (page) ())
(lisplet-register-page-location *test-lisplet* 'empty-page "index.html"
                                :welcome-page-p t)
```

This will provide an empty page at the URL "http://localhost:4242/test/index.html" and, since it is defined as a welcome page when you'll access the URL "http://localhost:4242/test" you will redirected to it.

4 Web application pages

CLAW applications are usually made of pages.

A PAGE is a CLOS class that contains the rendering logic and is called by the LISPLET when the URL matches the page resource mapping.

4.1 Writing your first CLAW page

You already know how to register a PAGE into a LISPLET, if not, visit the previous chapter; what you miss, is how to put content into a page.

CLAW comes with the full set of html tags plus some custom components that we'll see in the next sections.

All html tag are rendered with functions whose names are the tag name plus the character >. Tag attributes are pairs of symbols for attribute names and strings for their values.

For example the function that render a DIV tag with class attribute "foo" is:

```
(div> :class "foo")
```

Given this short intro we are now ready to write our first CLAW page:

```
(defclass index-page (page) ())
(defmethod page-content ((index-page index-page))
  (html>
   (head>
    (title> "First sample page"))
   (body>
    (h1> "Hello world"))))
(lisplet-register-page-location *test-lisplet* 'index-page "index.html"
                                :welcome-page-p t)
```

So, overriding the method PAGE-CONTENT for your new defined page gives you the possibility to insert its content. As you can see the method definition is very similar to an HTML file, thought more concise.

4.1.1 The special tag attribute: :ID and :STATIC-ID

CLAW pages try to keep "ID" tag attributes unique among the page. This is particularly useful when you have to render tags that expose their id inside a loop. To see what happens when this situation occurs see the following example:

```
(defmethod page-content ((sample-page sample-page))
  (html>
   (head>
    (title> "First sample page"))
   (body>
    (loop for letter in (list "A" "B" "C" "D")
       collect (div> :id "item" letter)))))
```

will produce the following HTML code

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>First sample page</title>
  </head>
  <body>
    <div id="item">A</div>
    <div id="item_1">B</div>
    <div id="item_2">C</div>
    <div id="item_3">D</div>
    <script type="text/javascript">
//<!--
document.addEventListener('DOMContentLoaded', function(e) {}, false);
//-->
    </script>
  </body>
</html>
```

When you want to prevent the default behaviour on id generation you have to provide the tag with the attribute `:STATIC-ID`, that will render into HTML as the attribute `:ID`, but without the id unique logic generation.

An important method that is present in all tags and component is the `GENERATE-ID` that you may use to obtain a unique id and put into components or tags with `:STATIC-ID`.

Look at the following to see how it can work:

```
(defmethod page-content ((sample-page sample-page))
  (html>
   (head>
    (title> "First sample page"))
   (body>
    (loop for letter in (list "A" "B" "C" "D")
       for id = (generate-id "item") then (generate-id "item")
       collect (div> (span> :static-id id letter)
     (span> :onclick
    (format nil "alert(document.getElementById('~a').innerHTML);"
    id)
    "click me"))))))
```

that will produce the following HTML code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>First sample page</title>
  </head>
  <body>
    <div>
      <span id="item">A</span>
      <span onclick="alert(document.getElementById('item').innerHTML);">
        click me</span>
    </div>
    <div>
      <span id="item_1">B</span>
      <span onclick="alert(document.getElementById('item_1').innerHTML);">
        click me</span>
    </div>
    <div>
      <span id="item_2">C</span>
      <span onclick="alert(document.getElementById('item_2').innerHTML);">
        click me</span>
    </div>
    <div>
      <span id="item_3">D</span>
      <span onclick="alert(document.getElementById('item_3').innerHTML);">
        click me</span>
    </div>
    <script type="text/javascript">
//<!--
document.addEventListener('DOMContentLoaded', function(e) {}, false);
//-->
    </script>
  </body>
</html>
```

So, the outside tag generated id, is used in the onclick method of the span tags to reference the previous tag.

5 Getting started with CLAW

6 Internationalization of our application

7 CLAW forms and form components

8 Input validation and field translations

9 Creating a web application by writing reusable components

10 Writing advanced components

11 Access validation and authorization

12 Advanced techniques

Function index

C

clawserver-address 4
clawserver-input-chunking-p 5
clawserver-mod-lisp-p 5
clawserver-name 4
clawserver-port 4
clawserver-read-timeout 5
clawserver-register-lisplet 7
clawserver-setgid 6
clawserver-setuid 6
clawserver-ssl-certificate-file 6
clawserver-ssl-privatekey-file 6
clawserver-ssl-privatekey-password 6
clawserver-sslname 4
clawserver-sslport 4
clawserver-start 7
clawserver-stop 7
clawserver-unregister-lisplet 7
clawserver-use-apache-log-p 5
clawserver-write-timeout 5

L

lisplet 7