NAME

/etc/yaws/yaws.conf − Configuration file for the Yaws web server

DESCRIPTION

Yaws is fast lightweight web server. It reads a configuration file called yaws.conf to control its operations. The configuration contains two distinct parts a global part which affects all the virtual hosts and a server part where options for each virtual host is supplied.

GLOBAL PART

logdir = Directory

All Yaws logs will be written to files in this directory. There are several different log files written by Yaws:

report.log - this is a text file that contains all error logger printouts from Yaws.

<Host>.access - for each virtual host served by Yaws, a file <Host>.access will be written which contains an access log in Common Log Format. (See http://en.wikipedia.org/wiki/Common_Log_Format for more details on Common Log Format.)

<Host>.auth - for each virtual host served by Yaws, a file <Host>.auth will be written which contains all http auth related messages.

trace_<YYYYMMDD_hhmmss> - Trace files are written in this subdirectory, suffixed by the creation date.

trace.<Pid>.http - this file contains the HTTP trace if that is enabled, where <Pid> is the process id handling the TCP connection.

trace.<Pid>.traffic - this file contains the traffic trace if that is enabled, where <Pid> is the process id handling the TCP connection.

Note that <Host>.access and <Host>.auth files will be used only if the directive **logger_mod** is not set or set to yaws_log. The default value for logdir is *"."*

ebin_dir = Directory

This directive adds Directory to the Erlang search path. It is possible to have several of these commands in the configuration file. The default value is *"yaws_dir"/examples/ebin*

id = String

It is possible run multiple Yaws servers on the same machine. We use the id of a Yaws server to control it using the different control commands such as:

```
# /usr/local/bin/yaws --id foobar --stop
```

To stop the Yaws server with id "foobar". Each Yaws server will write its internals data into a file called $HOME/.yaws/yaws/ID where ID is the identity of the server. Yaws also creates a file called ${VARDIR}/run/yaws/ctl-${ID} which contain the port number where the server is listening for control commands. The default id is *"default"*.

server_signature = String

This directive sets the "Server: " output header to the custom value. The default value is *"yaws/%VSN%, Yet Another Web Server"*.

include_dir = Directory

This directive adds Directory to the path of directories where the Erlang compiler searches for include files. We need to use this if we want to include .hrl files in our Yaws Erlang code. It is possible to have several of these commands in the configuration file. The default value is *"yaws_dir"/examples/include*.

max_num_cached_files = Integer

Yaws will cache small files such as commonly accessed GIF images in RAM. This directive sets a maximum number on the number of cached files. The default value is *400*.

max_num_cached_bytes = Integer

This directive controls the total amount of RAM which can maximally be used for cached RAM files. The default value is *1000000*, 1 megabyte.

max_size_cached_file = Integer

This directive sets a maximum size on the files that are RAM cached by Yaws. The default value is *8000*, 8 kBytes.

cache_refresh_secs = Integer

The RAM cache is used to serve pages that sit in the cache. An entry sits in cache at most cache_refresh_secs number of seconds. The default is *30*. This means that when the content is updated under the docroot, that change doesn't show until 30 seconds have passed. While developing a Yaws site, it may be convenient to set this value to 0. If the debug flag (-d) is passed to the Yaws start script, this value is automatically set to 0.

trace = false | traffic | http

This enables traffic or http tracing. Tracing is also possible to enable with a command line flag to Yaws. Default is *false*.

use_old_ssl = true | false

This re-enables the old OTP ssl implementation. By default we use the new ssl implementation.

auth_log = true | false

Deprecated and ignored. Now, this target must be set in server part.

max_connections = nolimit | Integer

Set this value to control the maximum number of connections from HTTP clients into the server. This is implemented by closing the last socket if the limit threshold is reached.

keepalive_maxuses = nolimit | Integer

Normally, Yaws does not restrict the number of times a connection is kept alive using keepalive. Setting this parameter to an integer X will ensure that connections are closed once they have been used X times. This can be a useful to guard against long running connections collecting too much garbage in the Erlang VM.

process_options = undefined | Proplist

Set process spawn options for client acceptor processes. Options must be specified as a quoted string of either the atom *undefined* or as a proplist of valid process options. The supported options are *fullsweep_after*, *min_heap_size*, and *min_bin_vheap_size*, each taking an associated integer value. Other process options are ignored. The proplist may also be empty. See **erlang:spawn_opt/4** for details on these options.

acceptor_pool_size = Integer

Set the size of the pool of cached acceptor processes. The specified value must be greater than or equal to 0. The default value is *8*. Specifying a value of 0 effectively disables the process pool.

log_wrap_size = Integer

The logs written by Yaws are all wrap logs, the default value at the size where they wrap around and the original gets renamed to File.old is *1000000*, 1 megabyte. This value can be changed.
If we set the value to 0 the logs will never wrap. If we want to use Yaws in combination with a more traditional log wrapper such as logrotate, set the size to 0 and Yaws will reopen the logfiles once they have be renamed/removed.

log_resolve_hostname = true | false

By default the client host IP is not resolved in the access logs.

fail_on_bind_err = true | false

Fail completely or not if Yaws fails to bind a listen socket Default is *true*.

enable_soap = true | false

If true, a soap server will be started at startup of Yaws. Default is *false*.

soap_srv_mods = ListOfModuleSetting

If enable_soap is true, a startup Yaws will invoke *yaws_soap_srv:setup()* to setup modules set here. ModuleSetting is either a triad like *<Mod, HandlerFunc, WsdlFile>* or a quadruple form like *<Mod, HandlerFunc, WsdlFile, Prefix>* which specifies the *prefix*. A *prefix* will be used as argument of *yaws_soap_lib:initModel()* and then be used as a XML namespace prefix. Note, the *WsdlFile* here should be an absolute-path file in local file systems.

For example, we can specify

```
soap_srv_mods=<Mod1, HandlerFunc, WsdlFile1> <Mod2, HandlerFunc, WsdlFile2, SpecifiedPrefix> ...
```

php_exe_path = Path

this target is deprecated and useless. use 'php_handler' target in server part instead.
The name of (and possibly path to) the php executable used to interpret php scripts (if allowed). Default is *php_exe_path = php-cgi*.

copy_error_log = true | false

Enable or disable copying of the error log. When we run in embedded mode, there may very well be some other systems process that is responsible for writing the errorlog to a file whereas when we run in normal standalone mode, we typically want the Erlang errorlog written to a report.log file. Default value is *true*.

ysession_mod = Module
Allows to specify a different Yaws session storage mechanism instead of an ETS table. One of the drawbacks of the default yaws_session_server implementation is that server side cookies are lost when the server restarts. Specifying a different module here will pass all writes/read operations to this module (it must implements appropriate callbacks).

runmod = ModuleName
At startup Yaws will invoke *ModuleName:start()* in a separate process. It is possible to have several runmods. This is useful if we want to reuse the Yaws startup shell script for our own application.

pick_first_virthost_on_nomatch = true | false
When Yaws gets a request, it extracts the Host: header from the client request to choose a virtual server amongst all servers with the same IP/Port pair. This configuration parameter decides whether Yaws should pick the first (as defined in the yaws.conf file) if no name match or not. In real live hosting scenarios we typically want this to be false whereas in testing/development scenarios it may be convenient to set it to true. Default is *true*.

keepalive_timeout = TimeInMilliseconds | infinity
If the HTTP session will be kept alive (i.e., not immediately closed) it will close after keepalive_timeout milliseconds unless a new request is received in that time. The default value is *30000*. The value *infinity* is legal but not recommended.

subconfig = File
Load specified config file.

subconfigdir = Directory
Load all config file in specified directory.

x_forwarded_for_log_proxy_whitelist = ListOfUpstreamProxyServerIps
this target is deprecated and will be ignored.

SERVER PART

Yaws can virthost several web servers on the same IP address as well as several web servers on different IP addresses. This includes SSL servers.

Each virtual host is defined within a matching pair of **<server ServerName>** and **</server>**. The ServerName will be the name of the webserver.

The following directives are allowed inside a server definition.

port = Port
This makes the server listen on Port. Default is *8000*.

listen = IpAddress
This makes the server listen on IpAddress. When virthosting several servers on the same ip/port address, if the browser doesn't send a Host: field, Yaws will pick the *first* server specified in the config file. If the specified IP address is 0.0.0.0 Yaws will listen on all local IP addresses on the specified port. Default is *0.0.0.0*. Multiple **listen** directives may be used to specify several addresses to listen on.

listen_backlog = Integer
This sets the TCP listen backlog for the server to define the maximum length the queue of pending connections may grow to. The default is the same as the default provided by *gen_tcp:listen/2*, which is 5.

server_signature = String
This directive sets the "Server: " output header to the custom value and overloads the global one for this virtual server.

rhost = Host[:Port]
This forces all local redirects issued by the server to go to Host. This is useful when Yaws listens to a port which is different from the port that the user connects to. For example, running Yaws as a non-privileged user makes it impossible to listen to port 80, since that port can only be opened by a privileged user. Instead Yaws listens to a high port number port, 8000, and iptables are used to redirect traffic to port 80 to port 8000 (most NAT:ing firewalls will also do this for you).

rscheme = http | https
This forces all local redirects issued by the server to use this method. This is useful when an SSL off-loader, or stunnel, is used in front of Yaws.

auth_log = true | false
Enable or disable the auth log for this virtual server. Default is *true*.

access_log = true | false
Setting this directive to false turns of traffic logging for this virtual server. The default value is *true*.

logger_mod = Module
It is possible to set a special module that handles access and auth logging. The default is to log all web server traffic to <Host>.access and <Host>.auth files in the configured or default logdir.
This module must implement the behaviour *yaws_logger*. Default value is *yaws_log*.

The following functions should be exported:

Module:open_log(ServerName, Type, LogDir)
When Yaws is started, this function is called for this virtual server. If the initialization is successful, the function must return *{true,State}* and if an error occurred, it must return *false*.

Module:close_log(ServerName, Type, State)
This function is called for this virtual server when Yaws is stopped.

Module:wrap_log(ServerName, Type, State, LogWrapSize)
This function is used to rotate log files. It is regularly called by Yaws and must return the possibly updated internal NewState.

Module:write_log(ServerName, Type, State, Infos)
When it needs to log a message, Yaws will call this function. The parameter Infos is

{Ip,Req,InHdrs,OutHdrs,Time} for an access log and *{Ip,Path,Item}* for an auth log, where:

Ip - IP address of the accessing client (as a tuple).

Req - the HTTP method, URI path, and HTTP version of the request (as a #http_request{} record).

InHdrs - the HTTP headers which were received from the WWW client (as a #headers{} record).

OutHdrs - the HTTP headers sent to the WWW client (as a #outh{} record)

Path - the URI path of the request (as a string).

Item - the result of an authentication request. May be *{ok,User}*, *403* or *{401,Realm}*.

Time - The time taken to serve the request, in microseconds.

For all of these callbacks, **ServerName** is the virtual server's name, *Type* is the atom access or auth and *State* is the internal state of the logger.

shaper = Module

Defines a module to control access to this virtual server. Access can be controlled based on the IP address of the client. It is also possible to throttles HTTP requests based on the client's download rate. This module must implement the behaviour *yaws_shaper*.

There is no such module configured by default.

dir_listings = true | true_nozip | false

Setting this directive to false disallows the automatic dir listing feature of Yaws. A status code 403 Forbidden will be sent. Set to true_nozip to avoid the auto-generated all.zip entries. Default is *false*.

extra_cgi_vars = ...

Add additional CGI or FastCGI variables. For example:

```
<extra_cgi_vars dir='/path/to/some/scripts'>
  var = val
  ...
</extra_cgi_vars>
```

statistics = true | false

Turns on/off statistics gathering for a virtual server. Default is *false*.

fcgi_app_server = Host:Port

The hostname and TCP port number of a FastCGI application server. The TCP port number is not optional. There is no default value.

fcgi_trace_protocol = true | false

Enable or disable tracing of FastCGI protocol messages as info log messages. Disabled by default.

fcgi_log_app_error = true | false
Enable or disable logging of application error messages (output to stderr and non-zero exit value). Disabled by default.

deflate = true | false
Turns on or off deflate compression for a server. Default is *false*.

<deflate> ... </deflate>
This begins and ends the deflate compression configuration for this server. The following items are allowed within a matching pair of <deflate> and </deflate> delimiters.

min_compress_size = nolimit | Integer
Defines the smallest response size that will be compressed. If nolimit is not used, the specified value must be strictly positive. The default value is *nolimit*.

compress_level = none | default | best_compression | best_speed | 0..9
Defines the compression level to be used. 0 (none), gives no compression at all, 1 (best_speed) gives best speed and 9 (best_compression) gives best compression. The default value is *default*.

window_size = 9..15
Specifies the zlib compression window size. It should be in the range 9 through 15. Larger values of this parameter result in better compression at the expense of memory usage. The default value is *15*.

mem_level = 1..9
Specifies how much memory should be allocated for the internal compression state. *mem_level=1* uses minimum memory but is slow and reduces compression ratio; *mem_level=9* uses maximum memory for optimal speed. The default value is *8*.

strategy = default | filtered | huffman_only
This parameter is used to tune the compression algorithm. See **zlib(3erl)** for more details on the *strategy* parameter. The default value is *default*.

use_gzip_static = true | false
If true, Yaws will try to serve precompressed versions of static files. It will look for precompressed files in the same location as original files that end in ".gz". Only files that do not fit in the cache are concerned. The default value is *false*.

mime_types = ListOfTypes | defaults | all

Restricts the deflate compression to particular mime types. The special value *all* enable it for all types (It is a synonym of '*/*'). Mime types into *ListOfTypes* must have the form 'type/subtype' or 'type/*' (indicating all subtypes of that type). Here is an example:

```
mime_types = default image/*
mime_types = application/xml application/xhtml+xml application/rss+xml
```

By default, following mime types are compressed (if **deflate** is set to true): *text/*, *application/rtf*, *application/msword*, *application/pdf*, *application/x-dvi*, *application/javascript*, *application/x-javascript*. Multiple mime_types directive can be used.

docroot = Directory ...

This makes the server serve all its content from Directory.
It is possible to pass a space-separated list of directories as docroot. If this is the case, the various directories will be searched in order for the requested file. This also works with the ssi and yssi constructs where the full list of directories will be searched for files to ssi/yssi include. Multiple docroot directives can be used. You need at least one valid docroot, invalid docroots are skipped with their associated auth structures.

auth_skip_docroot = true | false

If true, the docroot will not be searched for *.yaws_auth* files. This is useful when the docroot is quite large and the time to search it is prohibitive when Yaws starts up. Defaults to *false*.

partial_post_size = Integer | nolimit

When a Yaws file receives large POSTs, the amount of data received in each chunk is determined by the this parameter. The default value is *10240*.

dav = true | false

Turns on the DAV protocol for this server. The dav support in Yaws is highly limited. If dav is turned on, .yaws processing of .yaws pages is turned off. Default is *false*. Setting it to nolimit is potentially dangerous. The socket read timeout is supplied by the keepalive_timeout setting. If the read is not done within the timeout, the POST will fail.

tilde_expand = true|false

If this value is set to false Yaws will never do tilde expansion. The default is *false*. tilde_expansion is the mechanism whereby a URL on the form http://www.foo.com/~username is changed into a request where the docroot for that particular request is set to the directory ~username/public_html/.

allowed_scripts = ListOfSuffixes

The allowed script types for this server. Recognized are 'yaws', 'cgi', 'fcgi', 'php'. Default is *allowed_scripts = yaws php cgi fcgi*.

Note: for fcgi scripts, the FastCGI application server is only called if a local file with the .fcgi extension exists. However, the contents of the local .fcgi file are ignored.

tilde_allowed_scripts = ListOfSuffixes

The allowed script types for this server when executing files in a users public_html folder Recognized are 'yaws', 'cgi', 'fcgi', 'php'. Default is *tilde_allowed_scripts* = i.e. empty

appmods = ListOfModuleNames

If any the names in ListOfModuleNames appear as components in the path for a request, the path request parsing will terminate and that module will be called. There is also an alternate syntax for specifying the appmods if we don't want our internal erlang module names to be exposed in the URL paths. We can specify

appmods = <Path1, Module1> <Path2, Modules2> ...

Assume for example that we have the URL http://www.hyber.org/myapp/foo/bar/baz?user=joe while we have the module foo defined as an appmod, the function foo:out(Arg) will be invoked instead of searching the filesystems below the point foo.
The Arg argument will have the missing path part supplied in its appmoddata field.

It is also possible to exclude certain directories from appmod processing. This is particulaly interesting for '/' appmods. Here is an example:

appmods = </, myapp exclude_paths icons js top/static>

The above configuration will invoke the 'myapp' erlang module on everything except any file found in directories, 'icons', 'js' and 'top/static' relative to the docroot.

errormod_404 = Module

It is possible to set a special module that handles 404 Not Found messages. The function *Module:out404(Arg, GC, SC)* will be invoked. The arguments are

Arg - a #arg{} record

GC - a #gconf{} record (defined in yaws.hrl)

SC - a #sconf{} record (defined in yaws.hrl)

The function can and must do the same things that a normal *out/1* does.

errormod_401 = Module

It is possible to set a special module that handles 401 Unauthorized messages. This can for example be used to display a login page instead. The function *Module:out401(Arg, Auth, Realm)* will be invoked. The arguments are

Arg - a #arg{} record

Auth - a #auth{} record

Realm - a string

The function can and must do the same things that a normal *out/1* does.

errormod_crash = Module

It is possible to set a special module that handles the HTML generation of server crash messages. The default is to display the entire formated crash message in the browser. This is good for debugging but not in production.

The function *Module:crashmsg(Arg, SC, Str)* will be called. The *Str* is the real crash message formated as a string.
The function must return, *{content,MimeType,Cont}* or *{html, Str}* or *{ehtml, Term}*. That data will be shipped to the client.

expires = ListOfExpires

Controls the setting of the *Expires* HTTP header and the *max-age* directive of the *Cache-Control* HTTP header in server responses for specific mime types. The expiration date can set to be relative to either the time the source file was last modified, or to the time of the client access. ListOfExpires is defined as follows:

expires = <MimeType1, access+Seconds> <MimeType2, modify+Seconds> ...

These HTTP headers are an instruction to the client about the document's validity and persistence. If cached, the document may be fetched from the cache rather than from the source until this time has passed. After that, the cache copy is considered "expired" and invalid, and a new copy must be obtained from the source. Here is an example:

expires = <image/gif, access+2592000> <image/png, access+2592000>
expires = <image/jpeg, access+2592000> <text/css, access+2592000>

arg_rewrite_mod = Module

It is possible to install a module that rewrites all the Arg #arg{} records at an early stage in the Yaws server. This can be used to do various things such as checking a cookie, rewriting paths etc.
The module *yaws_vdir* can be used in case you want to serve static content that is not located in your docroot. See the example at the bottom of this man page for how to use the *opaque + vdir* elements to instruct the *yaws_vdir* module what paths to rewrite.

start_mod = Module

Defines a user provided callback module. At startup of the server, Module:start/1 will be called. The #sconf{} record (defined in yaws.hrl) will be used as the input argument. This makes it possible for a user application to synchronize the startup with the Yaws server as well as getting hold of user specific configuration data, see the explanation for the <opaque> context.

revproxy = Prefix Url [intercept_mod Module]

Make Yaws a reverse proxy. *Prefix* is a path inside our own docroot and *Url* **argument is a URL pointing to a website we want to "mount" under the** *Prefix* path. This example:

revproxy = /tmp/foo http://yaws.hyber.org

makes the hyber website appear under */tmp/foo*.

It is possible to have multiple reverse proxies inside the same server.

You can optionally configure an interception module for each reverse proxy, allowing your application to examine and modify requests and HTTP headers as they pass through the proxy from client to backend server and also examine and modify responses and HTTP headers as they return from the backend server through the proxy to the client.

You specify an interception module by including the optional *intercept_mod* keyword followed by *Module*, which should be the name of your interception module.

An interception module is expected to export two functions: *rewrite_request/2* and

rewrite_response/2. The two arguments passed to *rewrite_request/2* function are a *#http_request{}* record and a *#headers{}* record, whereas *rewrite_response/2* function takes a *#http_response{}* record and also a *#headers{}* record. You can find definitions for these record types in the *yaws_api.hrl* header file. Each function can examine each record instance and can either return each original instance or can return a modified copy of each instance in its response. The *rewrite_request/2* function should return a tuple of the following form:

{ok, #http_request{}, #headers{}}

and the *rewrite_response/2* function should similarly return a tuple of the following form:

{ok, #http_response{}, #headers{}}

A *#headers{}* record can easily be manipulated in an interceptor using the functions listed below:

yaws_api:set_header/2, *yaws_api:set_header/3*
yaws_api:get_header/2, *yaws_api:get_header/3*
yaws_api:delete_header/2

Any failures in your interception module's functions will result in HTTP status code 500, indicating an internal server error.

fwdproxy = true|false

Make Yaws a forward proxy. By enabling this option you can use Yaws as a proxy for outgoing web traffic, typically by configuring the proxy settings in a web-browser to explicitly target Yaws as its proxy server.

servername = Name

If we're virthosting everal servers and want to force a server to match specific Host: headers we can do this with the "servername" directive. This name doesn't necessarily have to be the same as the the name inside <server Name> in certain NAT scenarios. Rarely used feature.

php_handler = <Type, Spec>

Set handler to interpret .php files. It can be one of the following definitions:

php_handler = <cgi, Filename> - The name of (and possibly path to) the php executable used to interpret php scripts (if allowed).

php_handler = <fcgi, Host:Port> - Use the specified fastcgi server to interpret .php files (if allowed).

Yaws does not start the PHP interpreter in fastcgi mode for you. To run PHP in fastcgi mode, call it with the -b option. For example:

```
php5-cgi -b '127.0.0.1:54321'
```

This starts a php5 in fastcgi mode listening on the local network interface. To make use of this PHP server from Yaws, specify:

```
php_handler = <fcgi, 127.0.0.1:54321>
```

The PHP interpreter needs read access to the files it is to serve. Thus, if you run it in a different security context than Yaws itself, make sure it has access to the .php files.

Please note that anyone who is able to connect to the php fastcgi server directly can use it to read any file to which it has read access. You should consider this when setting up a system with several mutually untrusted instances of php.

php_handler = <extern, Module:Function | Node:Module:Function> - Use an external handler, possibly on another node, to interpret .php files (if allowed).

To interpret a .php file, the function *Module:Function(Arg)* will be invoked (Evaluated inside a rpc call if a *Node* is specified), where Arg is a #arg{} record.
The function must do the same things that a normal out/1 does.

Default value is *<cgi, "/usr/bin/php-cgi">*.

phpfcgi = Host:Port

this target is deprecated. use 'php_handler' target in server part instead.
Use this directive is same as: php_handler = <fcgi, Host:Port>.

<ssl> ... </ssl>

This begins and ends an SSL configuration for this server. It's possible to virthost several SSL servers on the same IP given that they all share the same certificate configuration. In general it is complicated to virthost several SSL servers on the same IP address since the certificate is typically bound to a domainname in the common name part of the certificate. One solution (the only?) to this problem is to have a certificate with multiple subjectAltNames. See http://wiki.cacert.org/VhostTaskForce#Interoperability_Test

keyfile = File

Specifies which file contains the private key for the certificate. If not specified then the certificate file will be used.

certfile = File

Specifies which file contains the certificate for the server.

cacertfile = File

A file containing trusted certificates to use during client authentication and to use when attempting to build the server certificate chain. The list is also used in the list of acceptable client CAs passed to the client when a certificate is requested.

verify = 0 | 1 | 2 | verify_none | verify_peer

Specifies the level of verification the server does on client certs. 0 means that the server will not ask for a cert (verify_none), 1 means that the server will ask the client for a cert but not fail if the client does not supply a client cert (verify_peer, fail_if_no_peer_cert = false), 2 means that the server requires the client to supply a client cert (verify_peer, fail_if_no_peer_cert = true).

Setting verify_none means that the x509 validation will be skipped (no certificate request is sent to the client), verify_peer means that a certificate request is sent to the client (x509

validation is performed.

You might want to use fail_if_no_peer_cert in combination with verify_peer.

fail_if_no_peer_cert = true | false
If verify is set to verify_peer and set to true the connection will fail if the client does not send a certificate (i.e. an empty certificate). If set to false the server will fail only if an invalid certificate is supplied (an empty certificate is considered valid).

depth = Int
Specifies the depth of certificate chains the server is prepared to follow when verifying client certs. For the OTP new ssl implementation it is also used to specify how far the server, i.e. we, shall follow the SSL certificates we present to the clients. Hence, using self signed certs, we typically need to set this to 0.

password = String
String If the private key is encrypted on disc, this password is the 3Dee key to decrypt it.

ciphers = String
This string specifies the SSL cipher string. The syntax of the SSL cipher string is a little horrible sublanguage of its own. It is documented in the ssl man page for "ciphers".

<redirect> ... </redirect>
Defines a redirect mapping. The following items are allowed within a matching pair of <redirect> and </redirect> delimiters.

We can have a series of **Path = URL** or **Path = file**

All accesses to Path will be redirected to URL/Path or alternatively to scheme:host:port/file/Path if a file is used. Note that the original path is appended to the redirected url. So if we for example have:

```
<redirect>
  /foo = http://www.mysite.org/zapp
  /bar = /tomato.html
</redirect>
```

Asumming this config resides on a site called http://abc.com, We have the following redirects:

http://abc.com/foo -> http://www.mysite.org/zapp/foo

http://abc.com/foo/test -> http://www.mysite.org/zapp/foo/test

http://abc.com/bar -> http://abc.com/bar

http://abc.com/bar/x/y/z -> http://abc.com/bar/x/y/z

Sometimes we do not want to have the original path appended to the redirected path. To get that behaviour we specify the config with '==' instead of '='.

```
<redirect>
  /foo == http://www.mysite.org/zapp
  /bar = /tomato.html
</redirect>
```

Now a request for http://abc.com/foo/x/y/z simply gets redirected to http://www.mysite.org/zapp. This is typically used when we simply want a static redirect at some place in the docroot.

When we specify a file as target for the redirect, the redir will be to the current http(s) server.

<auth> ... </auth>

Defines an auth structure. The following items are allowed within a matching pair of <auth> and </auth> delimiters.

docroot = Docroot

If a docroot is defined, this auth structure will be tested only for requests in the specified docroot. No docroot configured means all docroots. If two auth structures are defined, one with a docroot and one with no docroot, the first of both overrides the second one for requests in the configured docroot.

dir = Dir

Makes Dir to be controlled by WWW-authenticate headers. In order for a user to have access to WWW-Authenticate controlled directory, the user must supply a password. The Dir must be specified relative to the docroot. Multiple dir can be used. If no dir is set, the default value, "/", will be used.

realm = Realm

In the directory defined here, the WWW-Authenticate Realm is set to this value.

authmod = AuthMod

If an auth module is defined then AuthMod:auth(Arg, Auth) will be called for all access to the directory. The auth/2 function should return one of: true, false, {false, Realm}, {appmod, Mod}. If {appmod, Mod} is returned then a call to Mod:out401(Arg, Auth, Realm) will be used to deliver the content. If errormod_401 is defined, the call to Mod will be ignored. (Mod:out(Arg) is deprecated).

This can, for example, be used to implement cookie authentication. The auth() callback would check if a valid cookie header is present, if not it would return {appmod, ?MODULE} and the out401/1 function in the same module would return {redirect_local, "/login.html"}.

user = User:Password

Inside this directory, the user User has access if the user supplies the password Password in the popup dialogue presented by the browser. We can obviously have several of these value

inside a single <auth> </auth> pair.

The usage of User:Password in the actual config file is deprecated as of release 1.51. It is preferred to have the users in a file called *.yaws_auth* in the actual directory. The .yaws_auth file has to be file parseable by *file:consult/1*

Each row of the file must contain terms on the form

{User, Password}.

Where both User and Password should be strings. The .yaws_auth file mechanism is recursive. Thus any subdirectories to Dir are automatically also protected.

The .yaws_auth file is never visible in a dir listing

pam service = *pam-service*

If the item **pam** is part of the auth structure, Yaws will also try to authenticate the user using "pam" using the pam *service* indicated. Usual services are typically found under /etc/pam.d. Usual values are "system-auth" etc.

pam authentication is performed by an Erlang port program which is typically installed as suid root by the Yaws install script.

allow = all | ListOfHost

The *allow* directive affects which hosts can access an area of the server. Access can be controlled by IP address or IP address range. If all is specified, then all hosts are allowed access, subject to the configuration of the *deny* and *order* directives. To allow only particular hosts or groups of hosts to access the server, the host can be specified in any of the following formats:

A full IP address

allow = 10.1.2.3
allow = 192.168.1.104, 192.168.1.205

A network/netmask pair

allow = 10.1.0.0/255.255.0.0

A network/nnn CIDR specification

allow = 10.1.0.0/16

deny = all | ListOfHost

This directive allows access to the server to be restricted based on IP address. The arguments for the *deny* directive are identical to the arguments for the *allow* directive.

order = Ordering

The *order* directive, along with *allow* and *deny* directives, controls a three-pass access control system. The first pass processes either all *allow* or all *deny* directives, as specified by the *order* directive. The second pass parses the rest of the directives (*deny* or *allow*). The third

pass applies to all requests which do not match either of the first two.

Ordering is one of (Default value is *deny,allow*):

allow,deny
First, all *allow* directives are evaluated; at least one must match, or the request is rejected. Next, *deny* directives are evaluated. If any matches, the request is rejected. Last, any requests which do not match an *allow* or a *deny* directive are denied by default.

deny,allow
First, all *deny* directives are evaluated; if any match, the request is denied unless it also matches an *allow* directive. Any requests which do not match any *allow* or *deny* directives are permitted.

<opaque> ... </opaque>
This begins and ends an opaque configuration context for this server, where 'Key = Value' directives can be specified. These directives are ignored by Yaws (hence the name opaque), but can be accessed as a list of tuples *{Key,Value}* stored in the #sconf.opaque record entry. See also the description of the *start_mod* directive.

This mechanism can be used to pass data from a surrounding application into the individual .yaws pages.

EXAMPLES

The following example defines a single server on port 80.

```
logdir = /var/log/yaws
<server www.mydomain.org>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www
</server>
```

And this example shows a similar setup but two web servers on the same IP address.

```
logdir = /var/log/yaws
<server www.mydomain.org>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www
</server>

<server www.funky.org>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www_funky_org
</server>
```

An example with www-authenticate and no access logging at all.

```
logdir = /var/log/yaws
<server www.mydomain.org>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www
   access_log = false
   <auth>
      dir = secret/dir1
      realm = foobar
      user = jonny:verysecretpwd
      user = benny:thequestion
      user = ronny:havinganamethatendswithy
  </auth>
</server>
```

An example specifying a user defined module to be called at startup, as well as some user specific configuration.

```
<server www.funky.org>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www_funky_org
   start_mod = btt
   <opaque>
        mydbdir = /tmp
        mylogdir = /tmp/log
   </opaque>
</server>
```

An example specifying the GSSAPI/SPNEGO module (authmod_gssapi) to be used for authentication. This module requires egssapi version 0.1˜pre2 or later available at http://www.hem.za.org/egssapi/.

The Kerberos5 keytab is specified as 'keytab = File' directive in opaque. This keytab should contain the keys of the HTTP service principal, 'HTTP/www.funky.org' in this example.

```
<server www.funky.org>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www_funky_org
   start_mod = authmod_gssapi
   <auth>
        authmod = authmod_gssapi
        dir = secret/dir1
   </auth>
   <opaque>
        keytab = /etc/yaws/http.keytab
   </opaque>
</server>
```

And finally a slightly more complex example with two servers on the same IP, and one SSL server on a different IP.

When there are more than one server on the same IP, and they have different names the server must be able to choose one of them if the client doesn't send a Host: header. Yaws will choose the first one defined in the conf file.

```
logdir = /var/log/yaws
max_num_cached_files = 8000
max_num_cached_bytes = 6000000

<server www.mydomain.org>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www
</server>

<server www.funky.org>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www_funky_org
</server>

<server www.funky.org>
   port = 443
   listen = 192.168.128.32
   docroot = /var/yaws/www_funky_org
   <ssl>
     keyfile = /etc/funky.key
     certfile = /etc/funky.cert
     password = gazonk
   </ssl>
</server>
```

Finally an example with virtual directories, vdirs.

```
<server server.domain>
   port = 80
   listen = 192.168.128.31
   docroot = /var/yaws/www
   arg_rewrite_mod = yaws_vdir
   <opaque>
      vdir = "/virtual1/ /usr/local/somewhere/notrelated/to/main/docroot"
      vdir = "/myapp/ /some/other/path can include/spaces"
      vdir = "/icons/  /usr/local/www/yaws/icons"
   </opaque>
</server>
```

The first defined vdir can then be accessed at or under http://server.domain/virtual1/ or http://server.domain/virtual1

AUTHOR

Written by Claes Wikstrom

SEE ALSO

yaws(1) **erl**(1)

Comment] Local Variables: Comment] mode: nroff Comment] End: