stuart-thackray Support configuration for tlsv1.3 History

18 contributors +6

3598 lines (3186 sloc) | 134 KB

```erlang
%%%----------------------------------------------------------------------
%%% File    : yaws_config.erl
%%% Author  : Claes Wikstrom <klacke@bluetail.com>
%%% Purpose :
%%% Created : 16 Jan 2002 by Claes Wikstrom <klacke@bluetail.com>
%%%----------------------------------------------------------------------

-module(yaws_config).
-author('klacke@bluetail.com').


-include("../include/yaws.hrl").
-include("../include/yaws_api.hrl").
-include("yaws_debug.hrl").

-include_lib("kernel/include/file.hrl").

-define(NEXTLINE, io_get_line(FD, '', [])).

-export([load/1,
         make_default_gconf/2, make_default_sconf/0, make_default_sconf/3,
         add_sconf/1,
         add_yaws_auth/1,
         add_yaws_soap_srv/1, add_yaws_soap_srv/2,
         load_mime_types_module/2,
         compile_and_load_src_dir/1,
         search_sconf/3, search_group/3,
         update_sconf/4, delete_sconf/3,
         eq_sconfs/2, soft_setconf/4, hard_setconf/2,
         can_hard_gc/2, can_soft_setconf/4,
         can_soft_gc/2, verify_upgrade_args/2, toks/2]).

%% where to look for yaws.conf
paths() ->
    case application:get_env(yaws, config) of
        undefined ->
            case yaws:getuid() of
                {ok, "0"} ->    %% root
                    [yaws_generated:etcdir() ++ "/yaws/yaws.conf"];
                _ -> %% developer
                    [filename:join([yaws:home(), "yaws.conf"]),
                     "./yaws.conf",
                     yaws_generated:etcdir() ++ "/yaws/yaws.conf"]
            end;
        {ok, File} ->
            [File]
    end.



%% load the config

load(E = #env{conf = false}) ->
    case yaws:first(fun(F) -> yaws:exists(F) end, paths()) of
        false ->
            {error, "Can't find any config file "};
        {ok, _, File} ->
            load(E#env{conf = {file, File}})
    end;
load(E) ->
    {file, File} = E#env.conf,
    error_logger:info_msg("Yaws: Using config file ~s~n", [File]),
    case file:open(File, [read, {encoding, E#env.encoding}]) of
        {ok, FD} ->
            GC = make_default_gconf(E#env.debug, E#env.id),
            GC1 = if E#env.traceoutput == undefined ->
                          GC;
                     true ->
                          ?gc_set_tty_trace(GC, E#env.traceoutput)
                  end,
            GC2 =  ?gc_set_debug(GC1, E#env.debug),
            GC3 = GC2#gconf{trace = E#env.trace},
            R = fload(FD, GC3),
            ?Debug("FLOAD(~s): ~p", [File, R]),
            case R of
                {ok, GC4, Cs} ->
                    yaws:mkdir(yaws:tmpdir()),
                    Cs1 = add_yaws_auth(Cs),
                    add_yaws_soap_srv(GC4),
                    validate_cs(GC4, Cs1);
                Err ->
                    Err
            end;
        Err ->
            {error, ?F("Can't open config file ~s: ~p", [File, Err])}
    end.


add_yaws_soap_srv(GC) when GC#gconf.enable_soap == true ->
    add_yaws_soap_srv(GC, true);
add_yaws_soap_srv(_GC) ->
    [].
add_yaws_soap_srv(GC, false) when GC#gconf.enable_soap == true ->
    [{yaws_soap_srv, {yaws_soap_srv, start_link, [GC#gconf.soap_srv_mods]},
      permanent, 5000, worker, [yaws_soap_srv]}];
add_yaws_soap_srv(GC, true) when GC#gconf.enable_soap == true ->
    Spec = add_yaws_soap_srv(GC, false),
    case whereis(yaws_soap_srv) of
        undefined ->
            spawn(fun() -> supervisor:start_child(yaws_sup, hd(Spec)) end);
        _ ->
            ok
    end,
    Spec;
add_yaws_soap_srv(_GC, _Start) ->
    [].


add_yaws_auth(#sconf{}=SC) ->
    SC#sconf{authdirs = setup_auth(SC)};
add_yaws_auth(SCs) ->
    [SC#sconf{authdirs = setup_auth(SC)} || SC <- SCs].


%% We search and setup www authenticate for each directory
%% specified as an auth directory or containing a .yaws_auth file.
%% These are merged with server conf.
setup_auth(#sconf{docroot = Docroot, xtra_docroots = XtraDocroots,
                  authdirs = Authdirs}=SC) ->
    [begin
         Authdirs1 = load_yaws_auth_from_docroot(D, ?sc_auth_skip_docroot(SC)),
         Authdirs2 = load_yaws_auth_from_authdirs(Authdirs, D, []),
         Authdirs3 = [A || A <- Authdirs1,
                           not lists:keymember(A#auth.dir,#auth.dir,Authdirs2)],
         Authdirs4 = ensure_auth_headers(Authdirs3 ++ Authdirs2),
         start_pam(Authdirs4),
         {D, Authdirs4}
     end || D <- [Docroot|XtraDocroots] ].


load_yaws_auth_from_docroot(_, true) ->
    [];
load_yaws_auth_from_docroot(undefined, _) ->
    [];
load_yaws_auth_from_docroot(Docroot, _) ->
    Fun = fun (Path, Acc) ->
                  %% Strip Docroot and then filename
                  SP  = string:sub_string(Path, length(Docroot)+1),
                  Dir = filename:dirname(SP),
                  A = #auth{docroot=Docroot, dir=Dir},
                  case catch load_yaws_auth_file(Path, A) of
                      {ok, L} -> L ++ Acc;
                      _Other  -> Acc
                  end
          end,
    filelib:fold_files(Docroot, "^.yaws_auth$", true, Fun, []).


load_yaws_auth_from_authdirs([], _, Acc) ->
    lists:reverse(Acc);
load_yaws_auth_from_authdirs([Auth = #auth{dir=Dir}| Rest], Docroot, Acc) ->
    if
        Auth#auth.docroot /= [] andalso Auth#auth.docroot /= Docroot ->
            load_yaws_auth_from_authdirs(Rest, Docroot, Acc);
        Auth#auth.docroot == [] ->
            Auth1 = Auth#auth{dir=filename:nativename(Dir)},
            F = fun(A) ->
                        (A#auth.docroot == Docroot andalso
                         A#auth.dir == Auth1#auth.dir)
                end,
            case lists:any(F, Acc) of
                true ->
                    load_yaws_auth_from_authdirs(Rest, Docroot, Acc);
                false ->
                    Acc1 = Acc ++ load_yaws_auth_from_authdir(Docroot, Auth1),
                    load_yaws_auth_from_authdirs(Rest, Docroot, Acc1)
            end;
        true -> %% #auth.docroot == Docroot
            Auth1 = Auth#auth{docroot=Docroot, dir=filename:nativename(Dir)},
            F = fun(A) ->
                        not (A#auth.docroot == [] andalso
                             A#auth.dir == Auth1#auth.dir)
                end,
            Acc1 = lists:filter(F, Acc),
            Acc2 = Acc1 ++ load_yaws_auth_from_authdir(Docroot, Auth1),
            load_yaws_auth_from_authdirs(Rest, Docroot, Acc2)
    end;
load_yaws_auth_from_authdirs([{Docroot, Auths}|_], Docroot, Acc) ->
    load_yaws_auth_from_authdirs(Auths, Docroot, Acc);
load_yaws_auth_from_authdirs([_| Rest], Docroot, Acc) ->
    load_yaws_auth_from_authdirs(Rest, Docroot, Acc).


load_yaws_auth_from_authdir(Docroot, Auth) ->
    Dir = case Auth#auth.dir of
              "/" ++ R -> R;
              _        -> Auth#auth.dir
          end,
    Path = filename:join([Docroot, Dir, ".yaws_auth"]),
    case catch load_yaws_auth_file(Path, Auth) of
        {ok, Auths} -> Auths;
        _           -> [Auth]
    end.


load_yaws_auth_file(Path, Auth) ->
    case file:consult(Path) of
        {ok, TermList} ->
            error_logger:info_msg("Reading .yaws_auth ~s~n", [Path]),
            parse_yaws_auth_file(TermList, Auth);
        {error, enoent} ->
            {error, enoent};
        Error ->
            error_logger:format("Bad .yaws_auth file ~s ~p~n", [Path, Error]),
            Error
    end.


ensure_auth_headers(Authdirs) ->
    [add_auth_headers(Auth) || Auth <- Authdirs].

add_auth_headers(Auth = #auth{headers = []}) ->
    %% Headers needs to be set
    Realm   = Auth#auth.realm,
    Headers = yaws:make_www_authenticate_header({realm, Realm}),
    Auth#auth{headers = Headers};
add_auth_headers(Auth) ->
    Auth.


start_pam([]) ->
    ok;
start_pam([#auth{pam = false}|T]) ->
    start_pam(T);
start_pam([A|T]) ->
    case whereis(yaws_pam) of
        undefined ->    % pam not started
            Spec = {yaws_pam, {yaws_pam, start_link,
                               [yaws:to_list(A#auth.pam),undefined,undefined]},
                    permanent, 5000, worker, [yaws_pam]},
            spawn(fun() -> supervisor:start_child(yaws_sup, Spec) end);
        _ ->
            start_pam(T)
    end.


parse_yaws_auth_file([], Auth=#auth{files=[]}) ->
    {ok, [Auth]};
parse_yaws_auth_file([], Auth=#auth{dir=Dir, files=Files}) ->
    {ok, [Auth#auth{dir=filename:join(Dir, F), files=[F]} || F <- Files]};

parse_yaws_auth_file([{realm, Realm}|T], Auth0) ->
    parse_yaws_auth_file(T, Auth0#auth{realm = Realm});

parse_yaws_auth_file([{pam, Pam}|T], Auth0)
  when is_atom(Pam) ->
    parse_yaws_auth_file(T, Auth0#auth{pam = Pam});

parse_yaws_auth_file([{authmod, Authmod0}|T], Auth0)
  when is_atom(Authmod0)->
    Headers = try
                  Authmod0:get_header() ++ Auth0#auth.headers
              catch
                  _:_ ->
                      error_logger:format("Failed to ~p:get_header() \n",
                                          [Authmod0]),
                      Auth0#auth.headers
              end,
    parse_yaws_auth_file(T, Auth0#auth{mod = Authmod0, headers = Headers});

parse_yaws_auth_file([{file, File}|T], Auth0) ->
    Files = case File of
                "/" ++ F -> [F|Auth0#auth.files];
                _        -> [File|Auth0#auth.files]
            end,
    parse_yaws_auth_file(T, Auth0#auth{files=Files});

parse_yaws_auth_file([{User, Password}|T], Auth0)
  when is_list(User), is_list(Password) ->
    Salt = crypto:strong_rand_bytes(32),
    Hash = crypto:hash(sha256, [Salt, Password]),
    Users = case lists:member({User, sha256, Salt, Hash}, Auth0#auth.users) of
                true  -> Auth0#auth.users;
                false -> [{User, sha256, Salt, Hash} | Auth0#auth.users]
            end,
    parse_yaws_auth_file(T, Auth0#auth{users = Users});

parse_yaws_auth_file([{User, Algo, B64Hash}|T], Auth0)
  when is_list(User), is_list(Algo), is_list(B64Hash) ->
    case parse_auth_user(User, Algo, "", B64Hash) of
        {ok, Res} ->
            Users = case lists:member(Res, Auth0#auth.users) of
                        true  -> Auth0#auth.users;
                        false -> [Res | Auth0#auth.users]
                    end,
            parse_yaws_auth_file(T, Auth0#auth{users = Users});
        {error, Reason} ->
            error_logger:format("Failed to parse user line ~p: ~p~n",
                                [{User, Algo, B64Hash}, Reason]),
            parse_yaws_auth_file(T, Auth0)
    end;

parse_yaws_auth_file([{User, Algo, B64Salt, B64Hash}|T], Auth0)
  when is_list(User), is_list(Algo), is_list(B64Salt), is_list(B64Hash) ->
    case parse_auth_user(User, Algo, B64Salt, B64Hash) of
        {ok, Res} ->
            Users = case lists:member(Res, Auth0#auth.users) of
                        true  -> Auth0#auth.users;
                        false -> [Res | Auth0#auth.users]
                    end,
            parse_yaws_auth_file(T, Auth0#auth{users = Users});
        {error, Reason} ->
            error_logger:format("Failed to parse user line ~p: ~p~n",
                                [{User, Algo, B64Hash, B64Hash}, Reason]),
            parse_yaws_auth_file(T, Auth0)
    end;

parse_yaws_auth_file([{allow, all}|T], Auth0) ->
    Auth1 = case Auth0#auth.acl of
                none    -> Auth0#auth{acl={all, [], deny_allow}};
                {_,D,O} -> Auth0#auth{acl={all, D, O}}
            end,
    parse_yaws_auth_file(T, Auth1);

parse_yaws_auth_file([{allow, IPs}|T], Auth0) when is_list(IPs) ->
    Auth1 = case Auth0#auth.acl of
                none ->
                    AllowIPs = parse_auth_ips(IPs, []),
                    Auth0#auth{acl={AllowIPs, [], deny_allow}};
                {all, _, _} ->
                    Auth0;
                {AllowIPs, DenyIPs, Order} ->
                    AllowIPs2 = parse_auth_ips(IPs, []) ++ AllowIPs,
                    Auth0#auth{acl={AllowIPs2, DenyIPs, Order}}
            end,
    parse_yaws_auth_file(T, Auth1);

parse_yaws_auth_file([{deny, all}|T], Auth0) ->
    Auth1 = case Auth0#auth.acl of
                none    -> Auth0#auth{acl={[], all, deny_allow}};
                {A,_,O} -> Auth0#auth{acl={A, all, O}}
            end,
    parse_yaws_auth_file(T, Auth1);

parse_yaws_auth_file([{deny, IPs}|T], Auth0) when is_list(IPs) ->
    Auth1 = case Auth0#auth.acl of
                none ->
                    DenyIPs = parse_auth_ips(IPs, []),
                    Auth0#auth{acl={[], DenyIPs, deny_allow}};
                {_, all, _} ->
                    Auth0;
                {AllowIPs, DenyIPs, Order} ->
                    DenyIPs2 = parse_auth_ips(IPs, []) ++ DenyIPs,
                    Auth0#auth{acl={AllowIPs, DenyIPs2, Order}}
            end,
    parse_yaws_auth_file(T, Auth1);

parse_yaws_auth_file([{order, O}|T], Auth0)
  when O == allow_deny; O == deny_allow ->
    Auth1 = case Auth0#auth.acl of
                none    -> Auth0#auth{acl={[], [], O}};
                {A,D,_} -> Auth0#auth{acl={A, D, O}}
            end,
    parse_yaws_auth_file(T, Auth1).



%% Create mime_types.erl, compile it and load it. If everything is ok,
%% reload groups.
%%
%% If an error occured, the previously-loaded version (the first time, it's the
%% static version) is kept.
load_mime_types_module(GC, Groups) ->
    GInfo  = GC#gconf.mime_types_info,
    SInfos = [{{SC#sconf.servername, SC#sconf.port}, SC#sconf.mime_types_info}
              || SC <- lists:flatten(Groups),
                 SC#sconf.mime_types_info /= undefined],

    case {is_dir(yaws:id_dir(GC#gconf.id)), is_dir(yaws:tmpdir("/tmp"))} of
        {true, _} ->
            File = filename:join(yaws:id_dir(GC#gconf.id), "mime_types.erl"),
            load_mime_types_module(File, GInfo, SInfos);
```

```erlang
        {_, true} ->
            File = filename:join(yaws:tmpdir("/tmp"), "mime_types.erl"),
            load_mime_types_module(File, GInfo, SInfos);
        _ ->
            error_logger:format("Cannot write module mime_types.erl~n"
                                "Keep the previously-loaded version~n", [])
    end,
    lists:map(fun(Gp) ->
                      [begin
                           F   = fun(X) when is_atom(X) -> X;
                                    (X) -> element(1, mime_types:t(SC, X))
                                 end,
                           TAS = SC#sconf.tilde_allowed_scripts,
                           AS  = SC#sconf.allowed_scripts,
                           SC#sconf{tilde_allowed_scripts=lists:map(F, TAS),
                                    allowed_scripts=lists:map(F, AS)}
                       end || SC <- Gp]
              end, Groups).

load_mime_types_module(_, undefined, []) ->
    ok;
load_mime_types_module(File, undefined, SInfos) ->
    load_mime_types_module(File, #mime_types_info{}, SInfos);
load_mime_types_module(File, GInfo, SInfos) ->
    case mime_type_c:generate(File, GInfo, SInfos) of
        ok ->
            case compile:file(File, [binary]) of
                {ok, ModName, Binary} ->
                    case code:load_binary(ModName, [], Binary) of
                        {module, ModName} ->
                            ok;
                        {error, What} ->
                            error_logger:format(
                              "Cannot load module '~p': ~p~n"
                              "Keep the previously-loaded version~n",
                              [ModName, What]
                             )
                    end;
                _ ->
                    error_logger:format("Compilation of '~p' failed~n"
                                        "Keep the previously-loaded version~n",
                                        [File])
            end;
        {error, Reason} ->
            error_logger:format("Cannot write module ~p: ~p~n"
                                "Keep the previously-loaded version~n",
                                [File, Reason])
    end.


%% Compile modules found in the configured source directories, recursively.
compile_and_load_src_dir(GC) ->
    Incs = lists:map(fun(Dir) -> {i, Dir} end, GC#gconf.include_dir),
    Opts = [binary, return] ++ Incs,
    lists:foreach(fun(D) -> compile_and_load_src_dir([], [D], Opts) end,
                  GC#gconf.src_dir).

compile_and_load_src_dir(_Dir, [], _Opts) ->
    ok;
compile_and_load_src_dir(Dir, [Entry0|Rest], Opts) ->
    Entry1 = case Dir of
                 [] -> Entry0;
                 _  -> filename:join(Dir, Entry0)
             end,
    case filelib:is_dir(Entry1) of
        true ->
            case file:list_dir(Entry1) of
                {ok, Files} ->
                    compile_and_load_src_dir(Entry1, Files, Opts);
                {error, Reason} ->
                    error_logger:format("Failed to compile modules in ~p: ~s~n",
                                        [Entry1, file:format_error(Reason)])
            end;
        false ->
            case filename:extension(Entry0) of
                ".erl" -> compile_module_src_dir(Entry1, Opts);
                _      -> ok
            end
    end,
    compile_and_load_src_dir(Dir, Rest, Opts).


compile_module_src_dir(File, Opts) ->
    case catch compile:file(File, Opts) of
        {ok, Mod, Bin} ->
            error_logger:info_msg("Compiled ~p~n", [File]),
            load_src_dir(File, Mod, Bin);
        {ok, Mod, Bin, []} ->
            error_logger:info_msg("Compiled ~p [0 Errors - 0 Warnings]~n", [File]),
            load_src_dir(File, Mod, Bin);
        {ok, Mod, Bin, Warnings} ->
            WsMsg = [format_compile_warns(W,[]) || W <- Warnings],
            error_logger:warning_msg("Compiled ~p [~p Errors - ~p Warnings]~n~s",
                                     [File,0,length(WsMsg),WsMsg]),
            load_src_dir(File, Mod, Bin);
        {error, [], Warnings} ->
            WsMsg = [format_compile_warns(W,[]) || W <- Warnings],
            error_logger:format("Failed to compile ~p "
                                  "[~p Errors - ~p Warnings]~n~s"
                                  "*** warnings being treated as errors~n",
                                  [File,0,length(WsMsg),WsMsg]);
        {error, Errors, Warnings} ->
            WsMsg = [format_compile_warns(W,[]) || W <- Warnings],
            EsMsg = [format_compile_errs(E,[])  || E <- Errors],
            error_logger:format("Failed to compile ~p "
                                  "[~p Errors - ~p Warnings]~n~s~s",
                                  [File,length(EsMsg),length(WsMsg),EsMsg,WsMsg]);
        error ->
            error_logger:format("Failed to compile ~p~n", [File]);
        {'EXIT', Reason} ->
            error_logger:format("Failed to compile ~p: ~p~n", [File, Reason])
    end.


load_src_dir(File, Mod, Bin) ->
    case code:load_binary(Mod, File, Bin) of
        {module, Mod}   -> ok;
        {error, Reason} -> error_logger:format("Cannot load module ~p: ~p~n",
                                               [Mod, Reason])
    end.

format_compile_warns({_, []}, Acc) ->
    lists:reverse(Acc);
format_compile_warns({File, [{L,M,E}|Rest]}, Acc) ->
    Msg = io_lib:format("    ~s:~w: Warning: ~s~n", [File,L,M:format_error(E)]),
    format_compile_warns({File, Rest}, [Msg|Acc]).

format_compile_errs({_, []}, Acc) ->
    lists:reverse(Acc);
format_compile_errs({File, [{L,M,E}|Rest]}, Acc) ->
    Msg = io_lib:format("    ~s:~w: ~s~n", [File,L,M:format_error(E)]),
    format_compile_errs({File, Rest}, [Msg|Acc]).



%% This is the function that arranges sconfs into
%% different server groups
validate_cs(GC, Cs) ->
    L = lists:map(fun(#sconf{listen=IP0}=SC0) ->
                          SC = case is_tuple(IP0) of
                                   false ->
                                       {ok, IP} = inet_parse:address(IP0),
                                       SC0#sconf{listen=IP};
                                   true ->
                                       SC0
                               end,
                          {{SC#sconf.listen, SC#sconf.port}, SC}
                  end, Cs),
    L2 = lists:map(fun(X) -> element(2, X) end, lists:keysort(1,L)),
    L3 = arrange(L2, start, [], []),
    case validate_groups(GC, L3) of
        ok ->
            {ok, GC, L3};
        Err ->
            Err
    end.


validate_groups(_, []) ->
    ok;
validate_groups(GC, [H|T]) ->
    case (catch validate_group(GC, H)) of
        ok ->
            validate_groups(GC, T);
        Err ->
            Err
    end.

validate_group(GC, List) ->
    [SC0|SCs] = List,

    %% all servers with the same IP/Port must share the same tcp configuration
    case lists:all(fun(SC) ->
                           lists:keyfind(listen_opts, 1, SC#sconf.soptions) ==
                               lists:keyfind(listen_opts, 1, SC0#sconf.soptions)
                   end, SCs) of
        true ->
            ok;
        false ->
            throw({error, ?F("Servers in the same group must share the same tcp"
                                 " configuration: ~p", [SC0#sconf.servername])})
    end,

    %% If the default servers (the first one) is not an SSL server:
    %%    all servers  with the same IP/Port must be non-SSL server
    %% If SNI is disabled or not supported:
    %%    all servers with the same IP/Port must share the same SSL config
    %% If SNI is enabled:
    %%    TLS protocol must be supported by the default servers (the first one)
    if
        SC0#sconf.ssl == undefined ->
            case lists:all(fun(SC) -> SC#sconf.ssl == SC0#sconf.ssl end, SCs) of
                true  -> ok;
                false ->
                    throw({error, ?F("All servers in the same group than"
                                     " ~p must have no SSL configuration",
                                     [SC0#sconf.servername])})
            end;
        GC#gconf.sni == disable ->
            case lists:all(fun(SC) -> SC#sconf.ssl == SC0#sconf.ssl end, SCs) of
                true  -> ok;
                false ->
                    throw({error, ?F("SNI is disabled, all servers in the same"
                                     " group than ~p must share the same ssl"
                                     " configuration",
                                     [SC0#sconf.servername])})
            end;

        true ->
            Vs = case (SC0#sconf.ssl)#ssl.protocol_version of
                     undefined -> proplists:get_value(available,ssl:versions());
                     L         -> L
                 end,
            F = fun(V) -> lists:member(V, ['tlsv1.3','tlsv1.2','tlsv1.1',tlsv1]) end,
            case lists:any(F, Vs) of
                true -> ok;
                false ->
                    throw({error, ?F("SNI is enabled, the server ~p must enable"
                                     " TLS protocol", [SC0#sconf.servername])})
            end
    end,

    %% all servernames in a group must be unique
    SN = lists:sort([yaws:to_lower(X#sconf.servername) || X <- List]),
    no_two_same(SN).

no_two_same([H,H|_]) ->
    throw({error,
           ?F("Two servers in the same group cannot have same name ~p",[H])});
no_two_same([_H|T]) ->
    no_two_same(T);
no_two_same([]) ->
    ok.



arrange([C|Tail], start, [], B) ->
    C1 = set_server(C),
    arrange(Tail, {in, C1}, [C1], B);
arrange([], _, [], B) ->
    B;
arrange([], _, A, B) ->
    [lists:reverse(A) | B];
arrange([C|Tail], {in, C0}, A, B) ->
    C1 = set_server(C),
    if
        C1#sconf.listen == C0#sconf.listen,
        C1#sconf.port == C0#sconf.port ->
            arrange(Tail, {in, C0}, [C1|A], B);
        true ->
            arrange(Tail, {in, C1}, [C1], [lists:reverse(A)|B])
    end.


set_server(SC) ->
    SC1 = if
              SC#sconf.port == 0 ->
                  {ok, P} = yaws:find_private_port(),
                  SC#sconf{port=P};
              true ->
                  SC
          end,
    case {SC1#sconf.ssl, SC1#sconf.port, ?sc_has_add_port(SC1)} of
        {undefined, 80, _} ->
            SC1;
        {undefined, Port, true} ->
            add_port(SC1, Port);
        {_SSL, 443, _} ->
            SC1;
        {_SSL, Port, true} ->
            add_port(SC1, Port);
        {_,_,_} ->
            SC1
    end.


add_port(SC, Port) ->
    case string:tokens(SC#sconf.servername, ":") of
        [Srv, Prt] ->
            case (catch list_to_integer(Prt)) of
                {'EXIT', _} ->
                    SC#sconf{servername =
                                 Srv ++ [$:|integer_to_list(Port)]};
                _Int ->
                    SC
            end;
        [Srv] ->
            SC#sconf{servername =   Srv ++ [$:|integer_to_list(Port)]}
    end.


make_default_gconf(Debug, Id) ->
    Y = yaws_dir(),
    Flags = (?GC_COPY_ERRLOG bor ?GC_FAIL_ON_BIND_ERR bor
                 ?GC_PICK_FIRST_VIRTHOST_ON_NOMATCH),
    #gconf{yaws_dir = Y,
           ebin_dir = [filename:join([Y, "examples/ebin"])],
           include_dir = [filename:join([Y, "examples/include"])],
           trace = false,
           logdir = ".",
           cache_refresh_secs = if
                                    Debug == true ->
                                        0;
                                    true ->
                                        30
                                end,
           flags = if Debug -> Flags bor ?GC_DEBUG;
                      true  -> Flags
                   end,

           yaws = "Yaws " ++ yaws_generated:version(),
           id = Id
          }.

%% Keep this function for backward compatibility. But no one is supposed to use
%% it (yaws_config is an internal module, its api is private).
make_default_sconf() ->
    make_default_sconf([], undefined, undefined).

make_default_sconf([], Servername, Port) ->
    make_default_sconf(filename:join([yaws_dir(), "www"]), Servername, Port);
make_default_sconf(DocRoot, undefined, Port) ->
    make_default_sconf(DocRoot, "localhost", Port);
make_default_sconf(DocRoot, Servername, undefined) ->
    make_default_sconf(DocRoot, Servername, 8000);
make_default_sconf(DocRoot, Servername, Port) ->
    AbsDocRoot = filename:absname(DocRoot),
    case is_dir(AbsDocRoot) of
        true ->
            set_server(#sconf{port=Port, servername=Servername,
                              listen={127,0,0,1},docroot=AbsDocRoot});
        false ->
            throw({error, ?F("Invalid docroot: directory ~s does not exist",
                             [AbsDocRoot])})
    end.


yaws_dir() ->
    yaws:get_app_dir().

string_to_host_and_port(String) ->
    HostPortRE = "^(?:\\[([^\\]]+)\\]|([^:]+)):([0-9]+)$",
    REOptions = [{capture, all_but_first, list}],
    case re:run(String, HostPortRE, REOptions) of
        {match, [IPv6, HostOrIPv4, Port]} ->
            case string:to_integer(Port) of
                {Integer, []} when Integer >= 0, Integer =< 65535 ->
                    case IPv6 of
                        "" -> {ok, HostOrIPv4, Integer};
                        _  -> {ok, IPv6, Integer}
                    end;
                _Else ->
                    {error, ?F("~p is not a valid port number", [Port])}
            end;
        nomatch ->
            {error, ?F("bad host and port specifier, expected HOST:PORT; "
                "use [IP]:PORT for IPv6 address", [])}
    end.

string_to_node_mod_fun(String) ->
    case string:tokens(String, ":") of
        [Node, Mod, Fun] ->
            {ok, list_to_atom(Node), list_to_atom(Mod), list_to_atom(Fun)};
        [Mod, Fun] ->
            {ok, list_to_atom(Mod), list_to_atom(Fun)};
        _ ->
            {error, ?F("bad external module specifier, "
                       "expected NODE:MODULE:FUNCTION or MODULE:FUNCTION", [])}
    end.



%% two states, global, server
fload(FD, GC) ->
    case catch fload(FD, GC, [], 1, ?NEXTLINE) of
        {ok, GC1, Cs} -> {ok, GC1, lists:reverse(Cs)};
        Err           -> Err
    end.


fload(FD, GC, Cs, _Lno, eof) ->
    file:close(FD),
    {ok, GC, Cs};

fload(FD, GC, Cs, Lno, Chars) ->
    case toks(Lno, Chars) of
        [] ->
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);

        ["subconfig", '=', Name] ->
            case subconfigfiles(FD, Name, Lno) of
                {ok, Files} ->
```

```
                    case fload_subconfigfiles(Files, global, GC, Cs) of
                        {ok, GC1, Cs1} ->
                            fload(FD, GC1, Cs1, Lno+1, ?NEXTLINE);
                        Err ->
                            Err
                    end;
                Err ->
                    Err
            end;

        ["subconfigdir", '=', Name] ->
            case subconfigdir(FD, Name, Lno) of
                {ok, Files} ->
                    case fload_subconfigfiles(Files, global, GC, Cs) of
                        {ok, GC1, Cs1} ->
                            fload(FD, GC1, Cs1, Lno+1, ?NEXTLINE);
                        Err ->
                            Err
                    end;
                Err ->
                    Err
            end;

        ["trace", '=', Bstr] when GC#gconf.trace == false ->
            case Bstr of
                "traffic" ->
                    fload(FD, GC#gconf{trace = {true, traffic}}, Cs,
                          Lno+1, ?NEXTLINE);
                "http" ->
                    fload(FD, GC#gconf{trace = {true, http}}, Cs,
                          Lno+1, ?NEXTLINE);
                "false" ->
                    fload(FD, GC#gconf{trace = false}, Cs, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect false|http|traffic at line ~w",[Lno])}
            end;
        ["trace", '=', _Bstr] ->
            %% don't overwrite setting from commandline
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);


        ["logdir", '=', Logdir] ->
            Dir = case Logdir of
                      "+" ++ D ->
                          D1 = filename:absname(D),
                          %% try to make the log directory if it doesn't exist
                          yaws:mkdir(D1),
                          D1;
                      _ ->
                          filename:absname(Logdir)
                  end,
            case is_dir(Dir) of
                true ->
                    put(logdir, Dir),
                    fload(FD, GC#gconf{logdir = Dir}, Cs, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect directory at line ~w (logdir ~s)",
                               [Lno, Dir])}
            end;

        ["ebin_dir", '=', Ebindir] ->
            Dir = filename:absname(Ebindir),
            case warn_dir("ebin_dir", Dir) of
                true ->
                    fload(FD, GC#gconf{ebin_dir = [Dir|GC#gconf.ebin_dir]}, Cs,
                          Lno+1, ?NEXTLINE);
                false ->
                    fload(FD, GC, Cs, Lno+1, ?NEXTLINE)
            end;

        ["src_dir", '=', Srcdir] ->
            Dir = filename:absname(Srcdir),
            case warn_dir("src_dir", Dir) of
                true ->
                    fload(FD, GC#gconf{src_dir = [Dir|GC#gconf.src_dir]}, Cs,
                          Lno+1, ?NEXTLINE);
                false ->
                    fload(FD, GC, Cs, Lno+1, ?NEXTLINE)
            end;

        ["runmod", '=', Mod0] ->
            Mod = list_to_atom(Mod0),
            fload(FD, GC#gconf{runmods = [Mod|GC#gconf.runmods]}, Cs,
                  Lno+1, ?NEXTLINE);

        ["enable_soap", '=', Bool] ->
            if (Bool == "true") ->
                    fload(FD, GC#gconf{enable_soap = true}, Cs,
                          Lno+1, ?NEXTLINE);
               true ->
                    fload(FD, GC#gconf{enable_soap = false}, Cs,
                          Lno+1, ?NEXTLINE)
            end;

        ["soap_srv_mods", '=' | SoapSrvMods] ->
            case parse_soap_srv_mods(SoapSrvMods, []) of
                {ok, L} ->
                    fload(FD, GC#gconf{soap_srv_mods = L}, Cs,
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["max_connections", '=', Int] ->
            case (catch list_to_integer(Int)) of
                I when is_integer(I) ->
                    fload(FD, GC#gconf{max_connections = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ when Int == "nolimit" ->
                    fload(FD, GC, Cs, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["process_options", '=', POpts] ->
            case parse_process_options(POpts) of
                {ok, ProcList} ->
                    fload(FD, GC#gconf{process_options=ProcList}, Cs,
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["large_file_chunk_size", '=', Int] ->
            case (catch list_to_integer(Int)) of
                I when is_integer(I) ->
                    fload(FD, GC#gconf{large_file_chunk_size = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["large_file_sendfile", '=', Method] ->
            case set_sendfile_flags(GC, Method) of
                {ok, GC1} ->
                    fload(FD, GC1, Cs, Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["acceptor_pool_size", '=', Int] ->
            case catch list_to_integer(Int) of
                I when is_integer(I), I >= 0 ->
                    fload(FD, GC#gconf{acceptor_pool_size = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer >= 0 at line ~w", [Lno])}
            end;

        ["log_wrap_size", '=', Int] ->
            case (catch list_to_integer(Int)) of
                I when is_integer(I) ->
                    fload(FD, GC#gconf{log_wrap_size = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["log_resolve_hostname", '=',  Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    fload(FD, ?gc_log_set_resolve_hostname(GC, Val), Cs,
                          Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["fail_on_bind_err", '=',  Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    fload(FD, ?gc_set_fail_on_bind_err(GC, Val), Cs,
                          Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;


        ["include_dir", '=', Incdir] ->
            Dir = filename:absname(Incdir),
            case warn_dir("include_dir", Dir) of
                true ->
                    fload(FD, GC#gconf{include_dir= [Dir|GC#gconf.include_dir]},
                          Cs, Lno+1, ?NEXTLINE);
                false ->
                    fload(FD, GC, Cs, Lno+1, ?NEXTLINE)

            end;

        ["mnesia_dir", '=', Mnesiadir] ->
            Dir = filename:absname(Mnesiadir),
            case is_dir(Dir) of
                true ->
                    put(mnesiadir, Dir),
                    fload(FD, GC#gconf{mnesia_dir = Dir}, Cs, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect mnesia directory at line ~w", [Lno])}
            end;

        ["tmpdir", '=', _TmpDir] ->
            %% ignore
            error_logger:format(
              "tmpdir in yaws.conf is no longer supported - ignoring\n",[]
             ),
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);

        ["keepalive_timeout", '=', Val] ->
            %% keep this bugger for backward compat for a while
            case (catch list_to_integer(Val)) of
                I when is_integer(I) ->
                    fload(FD, GC#gconf{keepalive_timeout = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ when Val == "infinity" ->
                    fload(FD, GC#gconf{keepalive_timeout = infinity}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["keepalive_maxuses", '=', Int] ->
            case (catch list_to_integer(Int)) of
                I when is_integer(I) ->
                    fload(FD, GC#gconf{keepalive_maxuses = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ when Int == "nolimit" ->
                    %% nolimit is the default
                    fload(FD, GC, Cs, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["php_exe_path", '=' , PhpPath] ->
            error_logger:format(
              "'php_exe_path' is deprecated, use 'php_handler' instead\n",
              []),
            case is_file(PhpPath) of
                true ->
                    fload(FD, GC#gconf{phpexe = PhpPath}, Cs, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect executable file at line ~w", [Lno])}
            end;

        ["read_timeout", '=', _Val] ->
            %% deprecated, don't use
            error_logger:format(
              "read_timeout in yaws.conf is no longer supported - ignoring\n",[]
             ),
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);

        ["max_num_cached_files", '=', Val] ->
            case (catch list_to_integer(Val)) of
                I when is_integer(I) ->
                    fload(FD, GC#gconf{max_num_cached_files = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;


        ["max_num_cached_bytes", '=', Val] ->
            case (catch list_to_integer(Val)) of
                I when is_integer(I) ->
                    fload(FD, GC#gconf{max_num_cached_bytes = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;


        ["max_size_cached_file", '=', Val] ->
            case (catch list_to_integer(Val)) of
                I when is_integer(I) ->
                    fload(FD, GC#gconf{max_size_cached_file = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["cache_refresh_secs", '=', Val] ->
            case (catch list_to_integer(Val)) of
                I when is_integer(I), I >= 0 ->
                    fload(FD, GC#gconf{cache_refresh_secs = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect 0 or positive integer at line ~w",[Lno])}
            end;


        ["copy_error_log", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    fload(FD, ?gc_set_copy_errlog(GC, Val), Cs,
                          Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;


        ["auth_log", '=', Bool] ->
            error_logger:format(
              "'auth_log' global variable is deprecated and ignored."
              " it is now a per-server variable", []),
            case is_bool(Bool) of
                {true, _Val} ->
                    fload(FD, GC, Cs, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["id", '=', String] when GC#gconf.id == undefined;
                                 GC#gconf.id == "default" ->
            fload(FD, GC#gconf{id=String}, Cs, Lno+1, ?NEXTLINE);
        ["id", '=', String]  ->
            error_logger:format("Ignoring 'id = ~p' setting at line ~p~n",
                                [String,Lno]),
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);

        ["pick_first_virthost_on_nomatch", '=',  Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    fload(FD, ?gc_set_pick_first_virthost_on_nomatch(GC,Val),
                          Cs, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["use_fdsrv", '=',  _Bool] ->
            %% feature removed
            error_logger:format(
              "use_fdsrv in yaws.conf is no longer supported - ignoring\n",[]
             ),
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);

        ["use_old_ssl", '=',  _Bool] ->
            %% feature removed
            error_logger:format(
              "use_old_ssl in yaws.conf is no longer supported - ignoring\n",[]
             ),
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);

        ["use_large_ssl_pool", '=',  _Bool] ->
            %% just ignore - not relevant any longer
            error_logger:format(
              "use_large_ssl_pool in yaws.conf is no longer supported"
              " - ignoring\n", []
             ),
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);

        ["x_forwarded_for_log_proxy_whitelist", '=' | _] ->
            error_logger:format(
              "x_forwarded_for_log_proxy_whitelist in yaws.conf is no longer"
              " supported - ignoring\n", []
             ),
            fload(FD, GC, Cs, Lno+1, ?NEXTLINE);

        ["ysession_mod", '=', Mod_str] ->
            Ysession_mod = list_to_atom(Mod_str),
            fload(FD, GC#gconf{ysession_mod = Ysession_mod}, Cs,
                  Lno+1, ?NEXTLINE);

        ["ysession_cookiegen", '=', Mod_str] ->
            Ysession_cookiegen = list_to_atom(Mod_str),
            fload(FD, GC#gconf{ysession_cookiegen = Ysession_cookiegen}, Cs,
                  Lno+1, ?NEXTLINE);

        ["ysession_idle_timeout", '=', YsessionIdle] ->
            case (catch list_to_integer(YsessionIdle)) of
                I when is_integer(I), I > 0 ->
                    fload(FD, GC#gconf{ysession_idle_timeout = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect positive integer at line ~w",[Lno])}
            end;

        ["ysession_long_timeout", '=', YsessionLong] ->
            case (catch list_to_integer(YsessionLong)) of
                I when is_integer(I), I > 0 ->
                    fload(FD, GC#gconf{ysession_long_timeout = I}, Cs,
                          Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect positive integer at line ~w",[Lno])}
            end;

        ["server_signature", '=', Signature] ->
            fload(FD, GC#gconf{yaws=Signature}, Cs, Lno+1, ?NEXTLINE);

        ["default_type", '=', MimeType] ->
            case parse_mime_types_info(default_type, MimeType,
                                       GC#gconf.mime_types_info,
                                       #mime_types_info{}) of
                {ok, Info} ->
                    fload(FD, GC#gconf{mime_types_info=Info}, Cs,
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["default_charset", '=', Charset] ->
            case parse_mime_types_info(default_charset, Charset,
                                       GC#gconf.mime_types_info,
                                       #mime_types_info{}) of
                {ok, Info} ->
                    fload(FD, GC#gconf{mime_types_info=Info}, Cs,
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["mime_types_file", '=', File] ->
            case parse_mime_types_info(mime_types_file, File,
                                       GC#gconf.mime_types_info,
                                       #mime_types_info{}) of
                {ok, Info} ->
                    fload(FD, GC#gconf{mime_types_info=Info}, Cs,
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["add_types", '=' | NewTypes] ->
            case parse_mime_types_info(add_types, NewTypes,
                                       GC#gconf.mime_types_info,
                                       #mime_types_info{}) of
                {ok, Info} ->
                    fload(FD, GC#gconf{mime_types_info=Info}, Cs,
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["add_charsets", '=' | NewCharsets] ->
            case parse_mime_types_info(add_charsets, NewCharsets,
                                       GC#gconf.mime_types_info,
                                       #mime_types_info{}) of
                {ok, Info} ->
                    fload(FD, GC#gconf{mime_types_info=Info}, Cs,
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["nslookup_pref", '=' | Pref] ->
            case parse_nslookup_pref(Pref) of
                {ok, Families} ->
                    fload(FD, GC#gconf{nslookup_pref = Families}, Cs,
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["sni", '=', Sni] ->
            if
                Sni == "disable" ->
                    fload(FD, GC#gconf{sni=disable}, Cs, Lno+1, ?NEXTLINE);

                Sni == "enable" orelse Sni == "strict" ->
                    case yaws_dynopts:have_ssl_sni() of
                        true ->
                            fload(FD, GC#gconf{sni=list_to_atom(Sni)}, Cs, Lno+1,
                                  ?NEXTLINE);
                        _ ->
                            error_logger:info_msg("Warning, sni option is not"
                                                  " supported at line ~w~n", [Lno]),
                            fload(FD, GC, Cs, Lno+1, ?NEXTLINE)
                    end;
                true ->
                    {error, ?F("Expect disable|enable|strict at line ~w",[Lno])}
            end;

        ['<', "server", Server, '>'] ->
            C = #sconf{servername = Server, listen = [],
                       php_handler = {cgi, GC#gconf.phpexe}},
            fload(FD, server, GC, C, Cs, Lno+1, ?NEXTLINE);

        [H|_] ->
            {error, ?F("Unexpected tokens ~p at line ~w", [H, Lno])};
        Err ->
            Err
    end.


fload(FD, server, _GC, _C, _Cs, Lno, eof) ->
    file:close(FD),
    {error, ?F("Unexpected end-of-file at line ~w", [Lno])};

fload(FD, server, GC, C, Cs, Lno, Chars) ->
    case fload(FD, server, GC, C, Lno, Chars) of
        {ok, _, _, Lno1, eof} ->
            {error, ?F("Unexpected end-of-file at line ~w", [Lno1])};
        {ok, GC1, C1, Lno1, ['<', "/server", '>']} ->
            HasDocroot =
```

```erlang
                case C1#sconf.docroot of
                    undefined ->
                        Tests =
                            [fun() ->
                                     lists:keymember("/", #proxy_cfg.prefix,
                                                     C1#sconf.revproxy)
                             end,
                             fun() ->
                                     lists:keymember("/", 1,
                                                     C1#sconf.redirect_map)
                             end,
                             fun() ->
                                     lists:foldl(fun(_, true) -> true;
                                                    ({"/", _}, _Acc) -> true;
                                                    (_, Acc) -> Acc
                                                 end, false, C1#sconf.appmods)
                             end,
                             fun() ->
                                     ?sc_forward_proxy(C1)
                             end],
                        lists:any(fun(T) -> T() end, Tests);
                    _ ->
                        true
                end,
            case HasDocroot of
                true ->
                    case C1#sconf.listen of
                        [] ->
                            C2 = C1#sconf{listen = {127,0,0,1}},
                            fload(FD, GC1, [C2|Cs], Lno1+1, ?NEXTLINE);
                        Ls ->
                            Cs1 = [C1#sconf{listen=L} || L <- Ls] ++ Cs,
                            fload(FD, GC1, Cs1, Lno1+1, ?NEXTLINE)
                    end;
                false ->
                    {error,
                     ?F("No valid docroot configured for virthost "
                        "'~s' (port: ~w)",
                        [C1#sconf.servername, C1#sconf.port])}
            end;
        Err ->
            Err
    end.

fload(FD, extra_response_headers, GC, C, Lno, Chars) ->
    case toks(Lno, Chars) of
        [] ->
            fload(FD, extra_response_headers, GC, C, Lno+1, ?NEXTLINE);

        ["extramod", '=', Mod] ->
            ExtraResponseHdrs = C#sconf.extra_response_headers,
            C1 = C#sconf{extra_response_headers = [{extramod, list_to_atom(Mod)}|
                                                   ExtraResponseHdrs]},
            fload(FD, extra_response_headers, GC, C1, Lno+1, ?NEXTLINE);

        ["add", Hdr, '=', Value] ->
            ExtraResponseHdrs = C#sconf.extra_response_headers,
            C1 = C#sconf{extra_response_headers = [{add,Hdr,Value}|
                                                   ExtraResponseHdrs]},
            fload(FD, extra_response_headers, GC, C1, Lno+1, ?NEXTLINE);

        ["always", "add", Hdr, '=', Value] ->
            ExtraResponseHdrs = C#sconf.extra_response_headers,
            C1 = C#sconf{extra_response_headers = [{always_add,Hdr,Value}|
                                                   ExtraResponseHdrs]},
            fload(FD, extra_response_headers, GC, C1, Lno+1, ?NEXTLINE);

        ["add", Hdr, '='| Value] ->
            StringVal = lists:flatten(
                          yaws:join_sep(
                            lists:map(fun(V) when is_atom(V) ->
                                               atom_to_list(V);
                                          (V) -> V
                                       end, Value), " ")),
            ExtraResponseHdrs = C#sconf.extra_response_headers,
            C1 = C#sconf{extra_response_headers = [{add,Hdr,StringVal}|
                                                   ExtraResponseHdrs]},
            fload(FD, extra_response_headers, GC, C1, Lno+1, ?NEXTLINE);

        ["always", "add", Hdr, '='| Value] ->
            StringVal = lists:flatten(
                          yaws:join_sep(
                            lists:map(fun(V) when is_atom(V) ->
                                               atom_to_list(V);
                                          (V) -> V
                                       end, Value), " ")),
            ExtraResponseHdrs = C#sconf.extra_response_headers,
            C1 = C#sconf{extra_response_headers = [{always_add,Hdr,StringVal}|
                                                   ExtraResponseHdrs]},
            fload(FD, extra_response_headers, GC, C1, Lno+1, ?NEXTLINE);

        ["erase", Hdr] ->
            ExtraResponseHdrs = C#sconf.extra_response_headers,
            C1 = C#sconf{extra_response_headers = [{erase,Hdr}|
                                                   ExtraResponseHdrs]},
            fload(FD, extra_response_headers, GC, C1, Lno+1, ?NEXTLINE);

        ['<', "/extra_response_headers", '>'] ->
```

```
            fload(FD, server, GC, C, Lno+1, ?NEXTLINE);

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;

fload(FD, server, GC, C, Lno, eof) ->
    file:close(FD),
    {ok, GC, C, Lno, eof};
fload(FD, _,  _GC, _C, Lno, eof) ->
    file:close(FD),
    {error, ?F("Unexpected end-of-file at line ~w", [Lno])};

fload(FD, server, GC, C, Lno, Chars) ->
    case toks(Lno, Chars) of
        [] ->
            fload(FD, server, GC, C, Lno+1, ?NEXTLINE);

        ["subconfig", '=', Name] ->
            case subconfigfiles(FD, Name, Lno) of
                {ok, Files} ->
                    case fload_subconfigfiles(Files, server, GC, C) of
                        {ok, GC1, C1} ->
                            fload(FD, server, GC1, C1, Lno+1, ?NEXTLINE);
                        Err ->
                            Err
                    end;
                Err ->
                    Err
            end;

        ["subconfigdir", '=', Name] ->
            case subconfigdir(FD, Name, Lno) of
                {ok, Files} ->
                    case fload_subconfigfiles(Files, server, GC, C) of
                        {ok, GC1, C1} ->
                            fload(FD, server, GC1, C1, Lno+1, ?NEXTLINE);
                        Err ->
                            Err
                    end;
                Err ->
                    Err
            end;

        ["server_signature", '=', Sig] ->
            fload(FD, server, GC, C#sconf{yaws=Sig}, Lno+1, ?NEXTLINE);

        ["access_log", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = ?sc_set_access_log(C, Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["auth_log", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = ?sc_set_auth_log(C, Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["logger_mod", '=', Module] ->
            C1 = C#sconf{logger_mod = list_to_atom(Module)},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["dir_listings", '=', StrVal] ->
            case StrVal of
                "true" ->
                    C1 = ?sc_set_dir_listings(C, true),
                    C2 = ?sc_set_dir_all_zip(C1, true),
                    C3 = C2#sconf{appmods = [ {"all.zip", yaws_ls},
                                              {"all.tgz", yaws_ls},
                                              {"all.tbz2", yaws_ls}|
                                              C2#sconf.appmods]},
                    fload(FD, server, GC, C3, Lno+1, ?NEXTLINE);
                "true_nozip" ->
                    C1 = ?sc_set_dir_listings(C, true),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                "false" ->
                    C1 = ?sc_set_dir_listings(C, false),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect true|true_nozip|false at line ~w",[Lno])}
            end;

        ["deflate", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = C#sconf{deflate_options=#deflate{}},
                    C2 = ?sc_set_deflate(C1, Val),
                    fload(FD, server, GC, C2, Lno+1, ?NEXTLINE);
                false ->
```

```
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["auth_skip_docroot",'=',Bool] ->
            case is_bool(Bool) of
                {true,Val} ->
                    C1 = ?sc_set_auth_skip_docroot(C, Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["dav", '=', Bool] ->
            case is_bool(Bool) of
                {true, true} ->
                    %% Ever since WebDAV support was moved into an appmod,
                    %% we must no longer set the dav flag in the
                    %% sconf. Always turn it off instead.
                    C1 = ?sc_set_dav(C, false),
                    Runmods = GC#gconf.runmods,
                    GC1 = case lists:member(yaws_runmod_lock, Runmods) of
                              false ->
                                  GC#gconf{runmods=[yaws_runmod_lock|Runmods]};
                              true ->
                                  GC
                          end,
                    DavAppmods = lists:keystore(yaws_appmod_dav, 2,
                                                C1#sconf.appmods,
                                                {"/",yaws_appmod_dav}),
                    C2 = C1#sconf{appmods=DavAppmods},
                    fload(FD, server, GC1, C2, Lno+1, ?NEXTLINE);
                {true,false} ->
                    C1 = ?sc_set_dav(C, false),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["port", '=', Val] ->
            case (catch list_to_integer(Val)) of
                I when is_integer(I) ->
                    C1 = C#sconf{port = I},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["rmethod", '=', Val] ->
            case Val of
                "http" ->
                    C1 = C#sconf{rmethod = Val},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                "https" ->
                    C1 = C#sconf{rmethod = Val},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect http or https at line ~w", [Lno])}
            end;

        ["rhost", '=', Val] ->
            C1 = C#sconf{rhost = Val},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["listen", '=', IP] ->
            case inet_parse:address(IP) of
                {error, _} ->
                    {error, ?F("Expect IP address at line ~w:", [Lno])};
                {ok,Addr} ->
                    Lstn = C#sconf.listen,
                    C1 = if
                             is_list(Lstn) ->
                                 case lists:member(Addr, Lstn) of
                                     false ->
                                         C#sconf{listen = [Addr|Lstn]};
                                     true ->
                                         C
                                 end;
                             true ->
                                 C#sconf{listen = [Addr, Lstn]}
                         end,
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE)
            end;

        ["listen_backlog", '=', Val] ->
            case (catch list_to_integer(Val)) of
                B when is_integer(B) ->
                    C1 = update_soptions(C, listen_opts, backlog, B),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["servername", '=', Name] ->
            C1 = ?sc_set_add_port((C#sconf{servername = Name}),false),
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["serveralias", '=' | Names] ->
            C1 = C#sconf{serveralias = Names ++ C#sconf.serveralias},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        [ '<', "listen_opts", '>'] ->
            fload(FD, listen_opts, GC, C, Lno+1, ?NEXTLINE);

        ["docroot", '=', Rootdir | XtraDirs] ->
            RootDirs = lists:map(fun(R) -> filename:absname(R) end,
                                 [Rootdir | XtraDirs]),
            case lists:filter(fun(R) -> not is_dir(R) end, RootDirs) of
                [] when C#sconf.docroot =:= undefined ->
                    C1 = C#sconf{docroot = hd(RootDirs),
                                 xtra_docroots = tl(RootDirs)},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                [] ->
                    XtraDocroots = RootDirs ++ C#sconf.xtra_docroots,
                    C1 = C#sconf{xtra_docroots = XtraDocroots},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                NoDirs ->
                    error_logger:info_msg("Warning, Skip invalid docroots"
                                          " at line ~w : ~s~n",
                                          [Lno, string:join(NoDirs, ", ")]),
                    case lists:subtract(RootDirs, NoDirs) of
                        [] ->
                            fload(FD, server, GC, C, Lno+1, ?NEXTLINE);
                        [H|T] when C#sconf.docroot =:= undefined ->
                            C1 = C#sconf{docroot = H, xtra_docroots = T},
                            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                        Ds ->
                            XtraDocroots = Ds ++ C#sconf.xtra_docroots,
                            C1 = C#sconf{xtra_docroots = XtraDocroots},
                            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE)
                    end
            end;

        ["partial_post_size",'=',Size] ->
            case Size of
                "nolimit" ->
                    C1 = C#sconf{partial_post_size = nolimit},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                Val ->
                    case (catch list_to_integer(Val)) of
                        I when is_integer(I) ->
                            C1 = C#sconf{partial_post_size = I},
                            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                        _ ->
                            {error,
                             ?F("Expect integer or 'nolimit' at line ~w",
                                [Lno])}
                    end
            end;

        ['<', "auth", '>'] ->
            C1 = C#sconf{authdirs=[#auth{}|C#sconf.authdirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ['<', "redirect", '>'] ->
            fload(FD, server_redirect, GC, C, Lno+1, ?NEXTLINE);

        ['<', "deflate", '>'] ->
            C1 = C#sconf{deflate_options=#deflate{mime_types=[]}},
            fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);

        ["default_server_on_this_ip", '=', _Bool] ->
            error_logger:format(
              "default_server_on_this_ip in yaws.conf is no longer"
              " supported - ignoring\n", []
             ),
            fload(FD, server, GC, C, Lno+1, ?NEXTLINE);

        [ '<', "ssl", '>'] ->
            ssl_start(),
            fload(FD, ssl, GC, C#sconf{ssl = #ssl{}}, Lno+1, ?NEXTLINE);

        ["appmods", '=' | AppMods] ->
            case parse_appmods(AppMods, []) of
                {ok, L} ->
                    C1 = C#sconf{appmods = L ++ C#sconf.appmods},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["dispatchmod", '=', DispatchMod] ->
            C1 = C#sconf{dispatch_mod = list_to_atom(DispatchMod)},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["expires", '=' | Expires] ->
            case parse_expires(Expires, []) of
                {ok, L} ->
                    C1 = C#sconf{expires = L ++ C#sconf.expires},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["errormod_404", '=' , Module] ->
            C1 = C#sconf{errormod_404 = list_to_atom(Module)},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
```

```erlang

        ["errormod_crash", '=', Module] ->
            C1 = C#sconf{errormod_crash = list_to_atom(Module)},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["errormod_401", '=' , Module] ->
            C1 = C#sconf{errormod_401 = list_to_atom(Module)},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["arg_rewrite_mod", '=', Module] ->
            C1 = C#sconf{arg_rewrite_mod = list_to_atom(Module)},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["tilde_expand", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = ?sc_set_tilde_expand(C,Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ['<', "opaque", '>'] ->
            fload(FD, opaque, GC, C, Lno+1, ?NEXTLINE);

        ["start_mod", '=' , Module] ->
            C1 = C#sconf{start_mod = list_to_atom(Module)},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ['<', "rss", '>'] ->
            erase(rss_id),
            put(rss, []),
            fload(FD, rss, GC, C, Lno+1, ?NEXTLINE);

        ["tilde_allowed_scripts", '=' | Suffixes] ->
            C1 = C#sconf{tilde_allowed_scripts=Suffixes},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["allowed_scripts", '=' | Suffixes] ->
            C1 = C#sconf{allowed_scripts=Suffixes},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        ["index_files", '=' | Files] ->
            case parse_index_files(Files) of
                ok ->
                    C1 = C#sconf{index_files = Files},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["revproxy", '=' | Tail] ->
            case parse_revproxy(Tail) of
                {ok, RevProxy} ->
                    C1 = C#sconf{revproxy = [RevProxy | C#sconf.revproxy]},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                {error, url} ->
                    {error, ?F("Bad url at line ~p",[Lno])};
                {error, syntax} ->
                    {error, ?F("Bad revproxy syntax at line ~p",[Lno])};
                Error ->
                    Error
            end;

        ["fwdproxy", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = ?sc_set_forward_proxy(C, Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ['<', "extra_cgi_vars", "dir", '=', Dir, '>'] ->
            C1 = C#sconf{extra_cgi_vars=[{Dir, []}|C#sconf.extra_cgi_vars]},
            fload(FD, extra_cgi_vars, GC, C1, Lno+1, ?NEXTLINE);

        ["statistics", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = ?sc_set_statistics(C, Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["fcgi_app_server", '=' | Val] ->
            HostPortSpec = case Val of
                [HPS]                    -> HPS;
                ['[', HSpec, ']', PSpec] -> "[" ++ HSpec ++ "]" ++ PSpec
            end,
            case string_to_host_and_port(HostPortSpec) of
                {ok, Host, Port} ->
                    C1 = C#sconf{fcgi_app_server = {Host, Port}},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                {error, Reason} ->
                    {error, ?F("Invalid fcgi_app_server ~p at line ~w: ~s",
                               [HostPortSpec, Lno, Reason])}
            end;

        ["fcgi_trace_protocol", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = ?sc_set_fcgi_trace_protocol(C, Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["fcgi_log_app_error", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = ?sc_set_fcgi_log_app_error(C, Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["phpfcgi", '=', HostPortSpec] ->
            error_logger:format(
              "'phpfcgi' is deprecated, use 'php_handler' instead\n", []),
            case string_to_host_and_port(HostPortSpec) of
                {ok, Host, Port} ->
                    C1 = C#sconf{php_handler = {fcgi, {Host, Port}}},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                {error, Reason} ->
                    {error,
                     ?F("Invalid php fcgi server ~p at line ~w: ~s",
                        [HostPortSpec, Lno, Reason])}
            end;

        ["php_handler", '=' | PhpMod] ->
            case parse_phpmod(PhpMod, GC#gconf.phpexe) of
                {ok, I} ->
                    C1 = C#sconf{php_handler = I},
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                {error, Reason} ->
                    {error,
                     ?F("Invalid php_handler configuration at line ~w: ~s",
                        [Lno, Reason])}
            end;

        ["shaper", '=', Module] ->
            C1 = C#sconf{shaper = list_to_atom(Module)},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);


        ["default_type", '=', MimeType] ->
            case parse_mime_types_info(default_type, MimeType,
                                       C#sconf.mime_types_info,
                                       GC#gconf.mime_types_info) of
                {ok, Info} ->
                    fload(FD, server, GC, C#sconf{mime_types_info=Info},
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["default_charset", '=', Charset] ->
            case parse_mime_types_info(default_charset, Charset,
                                       C#sconf.mime_types_info,
                                       GC#gconf.mime_types_info) of
                {ok, Info} ->
                    fload(FD, server, GC, C#sconf{mime_types_info=Info},
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["mime_types_file", '=', File] ->
            case parse_mime_types_info(mime_types_file, File,
                                       C#sconf.mime_types_info,
                                       GC#gconf.mime_types_info) of
                {ok, Info} ->
                    fload(FD, server, GC, C#sconf{mime_types_info=Info},
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["add_types", '=' | NewTypes] ->
            case parse_mime_types_info(add_types, NewTypes,
                                       C#sconf.mime_types_info,
                                       GC#gconf.mime_types_info) of
                {ok, Info} ->
                    fload(FD, server, GC, C#sconf{mime_types_info=Info},
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["add_charsets", '=' | NewCharsets] ->
            case parse_mime_types_info(add_charsets, NewCharsets,
                                       C#sconf.mime_types_info,
                                       GC#gconf.mime_types_info) of
                {ok, Info} ->
```

```
                    fload(FD, server, GC, C#sconf{mime_types_info=Info},
                          Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["strip_undefined_bindings", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = ?sc_set_strip_undef_bindings(C, Val),
                    fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ['<', "extra_response_headers", '>'] ->
            fload(FD, extra_response_headers, GC, C, Lno+1, ?NEXTLINE);

        ['<', "/server", '>'] ->
            {ok, GC, C, Lno, ['<', "/server", '>']};

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;


fload(FD, listen_opts, GC, C, Lno, Chars) ->
    case toks(Lno, Chars) of
        [] ->
            fload(FD, listen_opts, GC, C, Lno+1, ?NEXTLINE);

        ["buffer", '=', Int] ->
            case (catch list_to_integer(Int)) of
                B when is_integer(B) ->
                    C1 = update_soptions(C, listen_opts, buffer, B),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["delay_send", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = update_soptions(C, listen_opts, delay_send, Val),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["linger", '=', Val] ->
            case (catch list_to_integer(Val)) of
                I when is_integer(I) ->
                    C1 = update_soptions(C, listen_opts, linger, {true, I}),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                _ when Val == "false" ->
                    C1 = update_soptions(C, listen_opts, linger, {false, 0}),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer|false at line ~w", [Lno])}
            end;

        ["nodelay", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = update_soptions(C, listen_opts, nodelay, Val),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["priority", '=', Int] ->
            case (catch list_to_integer(Int)) of
                P when is_integer(P) ->
                    C1 = update_soptions(C, listen_opts, priority, P),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["sndbuf", '=', Int] ->
            case (catch list_to_integer(Int)) of
                I when is_integer(I) ->
                    C1 = update_soptions(C, listen_opts, sndbuf, I),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["recbuf", '=', Int] ->
            case (catch list_to_integer(Int)) of
                I when is_integer(I) ->
                    C1 = update_soptions(C, listen_opts, recbuf, I),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer at line ~w", [Lno])}
            end;

        ["send_timeout", '=', Val] ->
            case (catch list_to_integer(Val)) of
                I when is_integer(I) ->
                    C1 = update_soptions(C, listen_opts, send_timeout, I),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                _ when Val == "infinity" ->
                    C1 = update_soptions(C, listen_opts, send_timeout,
                                         infinity),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer|infinity at line ~w", [Lno])}
            end;

        ["send_timeout_close", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = update_soptions(C, listen_opts, send_timeout_close,
                                         Val),
                    fload(FD, listen_opts, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ['<', "/listen_opts", '>'] ->
            fload(FD, server, GC, C, Lno+1, ?NEXTLINE);

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;

fload(FD, ssl, GC, C, Lno, Chars) ->
    case toks(Lno, Chars) of
        [] ->
            fload(FD, ssl, GC, C, Lno+1, ?NEXTLINE);

        %% A bunch of ssl options

        ["keyfile", '=', Val] ->
            case is_file(Val) of
                true ->
                    C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{keyfile = Val}},
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect existing file at line ~w", [Lno])}
            end;

        ["certfile", '=', Val] ->
            case is_file(Val) of
                true ->
                    C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{certfile = Val}},
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect existing file at line ~w", [Lno])}
            end;

        ["cacertfile", '=', Val] ->
            case is_file(Val) of
                true ->
                    C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{cacertfile = Val}},
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect existing file at line ~w", [Lno])}
            end;

        ["dhfile", '=', Val] ->
            case is_file(Val) of
                true ->
                    C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{dhfile = Val}},
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect existing file at line ~w", [Lno])}
            end;

        ["verify", '=', Val0] ->
            Fail0 = (C#sconf.ssl)#ssl.fail_if_no_peer_cert,
            {Val, Fail} = try
                              case list_to_integer(Val0) of
                                  0 -> {verify_none, Fail0};
                                  1 -> {verify_peer, false};
                                  2 -> {verify_peer, true};
                                  _ -> {error, Fail0}
                              end
                          catch error:badarg ->
                                  case list_to_atom(Val0) of
                                      verify_none -> {verify_none, Fail0};
                                      verify_peer -> {verify_peer, Fail0};
                                      _           -> {error, Fail0}
                                  end
                          end,
            case Val of
                error ->
                    {error, ?F("Expect integer or verify_none, "
                               "verify_peer at line ~w", [Lno])};
                _ ->
                    SSL = (C#sconf.ssl)#ssl{verify=Val,
                                             fail_if_no_peer_cert=Fail},
                    C1 = C#sconf{ssl=SSL},
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE)
            end;

        ["fail_if_no_peer_cert", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{
                                      fail_if_no_peer_cert = Val}},
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["depth", '=', Val0] ->
            Val = (catch list_to_integer(Val0)),
            case lists:member(Val, [0, 1,2,3,4,5,6,7]) of
                true ->
                    C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{depth = Val}},
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer 0..7 at line ~w", [Lno])}
            end;

        ["password", '=', Val] ->
            C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{password = Val}},
            fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);

        ["ciphers", '=', Val] ->
            try
                L = str2term(Val),
                Ciphers = ssl:cipher_suites(),
                case check_ciphers(L, Ciphers) of
                    ok ->
                        C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{ciphers = L}},
                        fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                    Err ->
                        Err
                end
            catch _:_ ->
                    {error, ?F("Bad cipherspec at line ~w", [Lno])}
            end;
        ["eccs", '=', Val] ->
            try
                L = str2term(Val),
                Curves = ssl:eccs(),
                case check_eccs(L, Curves) of
                    ok ->
                        C1 = C#sconf{ssl = (C#sconf.ssl)#ssl{eccs = L}},
                        fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                    Err ->
                        Err
                end
            catch _:_ ->
                    {error, ?F("Bad elliptic curves at line ~w", [Lno])}
            end;
        ["secure_renegotiate", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = C#sconf{ssl=(C#sconf.ssl)#ssl{secure_renegotiate=Val}},
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ["client_renegotiation", '=', Bool] ->
            case yaws_dynopts:have_ssl_client_renegotiation() of
                true ->
                    case is_bool(Bool) of
                        {true, Val} ->
                            C1 = C#sconf{ssl=(C#sconf.ssl)#ssl{client_renegotiation=Val}},
                            fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                        false ->
                            {error, ?F("Expect true|false at line ~w", [Lno])}
                    end;
                _ ->
                    error_logger:info_msg("Warning, client_renegotiation SSL "
                                          "option is not supported "
                                          "at line ~w~n", [Lno]),
                    fload(FD, ssl, GC, C, Lno+1, ?NEXTLINE)
            end;

        ["honor_cipher_order", '=', Bool] ->
            case yaws_dynopts:have_ssl_honor_cipher_order() of
                true ->
                    case is_bool(Bool) of
                        {true, Val} ->
                            C2 = C#sconf{
                                   ssl=(C#sconf.ssl)#ssl{honor_cipher_order=Val}
                                  },
                            fload(FD, ssl, GC, C2, Lno+1, ?NEXTLINE);
                        false ->
                            {error, ?F("Expect true|false at line ~w", [Lno])}
                    end;
                _ ->
                    error_logger:info_msg("Warning, honor_cipher_order SSL "
                                          "option is not supported "
                                          "at line ~w~n", [Lno]),
                    fload(FD, ssl, GC, C, Lno+1, ?NEXTLINE)
            end;

        ["protocol_version", '=' | Vsns0] ->
            try
                Vsns = [list_to_existing_atom(V) || V <- Vsns0, not is_atom(V)],
                C1 = C#sconf{
                       ssl=(C#sconf.ssl)#ssl{protocol_version=Vsns}
                      },
                fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE)
            catch _:_ ->
                    {error, ?F("Bad ssl protocol_version at line ~w", [Lno])}
            end;

        ["require_sni", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    C1 = C#sconf{
                           ssl=(C#sconf.ssl)#ssl{require_sni=Val}
                          },
                    fload(FD, ssl, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ['<', "/ssl", '>'] ->
            fload(FD, server, GC, C, Lno+1, ?NEXTLINE);

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;

fload(FD, server_auth, GC, C, Lno, Chars) ->
    [Auth|AuthDirs] = C#sconf.authdirs,
    case toks(Lno, Chars) of
        [] ->
            fload(FD, server_auth, GC, C, Lno+1, ?NEXTLINE);

        ["docroot", '=', Docroot] ->
            Auth1 = Auth#auth{docroot = filename:absname(Docroot)},
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["dir", '=', Dir] ->
            case file:list_dir(Dir) of
                {ok,_} when Dir /= "/" ->
                    error_logger:info_msg("Warning, authdir must be set "
                                          "relative docroot ~n",[]);
                _ ->
                    ok
            end,
            Dir1 = yaws_api:path_norm(Dir),
            Auth1 = Auth#auth{dir = [Dir1 | Auth#auth.dir]},
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["realm", '=', Realm] ->
            Auth1 = Auth#auth{realm = Realm},
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["authmod", '=', Mod] ->
            Mod1 = list_to_atom(Mod),
            code:ensure_loaded(Mod1),
            %% Add the auth header for the mod
            H = try
                    Mod1:get_header() ++ Auth#auth.headers
                catch _:_ ->
                        error_logger:format("Failed to ~p:get_header() \n",
                                            [Mod1]),
                        Auth#auth.headers
                end,
            Auth1 = Auth#auth{mod = Mod1, headers = H},
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["user", '=', User] ->
            case parse_auth_user(User, Lno) of
                {Name, Algo, Salt, Hash} ->
                    Auth1 = Auth#auth{
                              users = [{Name, Algo, Salt, Hash}|Auth#auth.users]
                             },
                    C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
                    fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, Str}
            end;

        ["allow", '=', "all"] ->
            Auth1 = case Auth#auth.acl of
                        none    -> Auth#auth{acl={all, [], deny_allow}};
                        {_,D,O} -> Auth#auth{acl={all, D, O}}
                    end,
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["allow", '=' | IPs] ->
            Auth1 = case Auth#auth.acl of
                        none ->
                            AllowIPs = parse_auth_ips(IPs, []),
                            Auth#auth{acl={AllowIPs, [], deny_allow}};
                        {all, _, _} ->
                            Auth;
                        {AllowIPs, DenyIPs, Order} ->
                            AllowIPs1 = parse_auth_ips(IPs, []) ++ AllowIPs,
                            Auth#auth{acl={AllowIPs1, DenyIPs, Order}}
                    end,
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["deny", '=', "all"] ->
            Auth1 = case Auth#auth.acl of
                        none    -> Auth#auth{acl={[], all, deny_allow}};
                        {A,_,O} -> Auth#auth{acl={A, all, O}}
                    end,
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["deny", '=' | IPs] ->
            Auth1 = case Auth#auth.acl of
                        none ->
                            DenyIPs = parse_auth_ips(IPs, []),
                            Auth#auth{acl={[], DenyIPs, deny_allow}};
                        {_, all, _} ->
                            Auth;
                        {AllowIPs, DenyIPs, Order} ->
                            DenyIPs1 = parse_auth_ips(IPs, []) ++ DenyIPs,
                            Auth#auth{acl={AllowIPs, DenyIPs1, Order}}
                    end,
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["order", '=', "allow", ',', "deny"] ->
            Auth1 = case Auth#auth.acl of
                        none    -> Auth#auth{acl={[], [], allow_deny}};
                        {A,D,_} -> Auth#auth{acl={A, D, allow_deny}}
                    end,
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["order", '=', "deny", ',', "allow"] ->
            Auth1 = case Auth#auth.acl of
                        none    -> Auth#auth{acl={[], [], deny_allow}};
                        {A,D,_} -> Auth#auth{acl={A, D, deny_allow}}
                    end,
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ["pam", "service", '=', Serv] ->
            Auth1 = Auth#auth{pam=Serv},
            C1 = C#sconf{authdirs=[Auth1|AuthDirs]},
            fload(FD, server_auth, GC, C1, Lno+1, ?NEXTLINE);

        ['<', "/auth", '>'] ->
            Pam = Auth#auth.pam,
            Users = Auth#auth.users,
            Realm = Auth#auth.realm,
            Auth1 =  case {Pam, Users} of
                         {false, []} ->
                             Auth;
                         _ ->
                             H = Auth#auth.headers ++
                                 yaws:make_www_authenticate_header({realm, Realm}),
                             Auth#auth{headers = H}
                     end,
            AuthDirs1 = case Auth1#auth.dir of
                            [] -> [Auth1#auth{dir="/"}|AuthDirs];
                            Ds -> [Auth1#auth{dir=D} || D <- Ds] ++ AuthDirs
                        end,
            C1 = C#sconf{authdirs=AuthDirs1},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;

fload(FD, server_redirect, GC, C, Lno, Chars) ->
    RedirMap = C#sconf.redirect_map,
    case toks(Lno, Chars) of
        [] ->
            fload(FD, server_redirect, GC, C, Lno+1, ?NEXTLINE);

        [Path, '=', '=' | Rest] ->
            %% "Normalize" Path
            Path1 = filename:join([yaws_api:path_norm(Path)]),
            case parse_redirect(Path1, Rest, noappend, Lno) of
                {error, Str} ->
                    {error, Str};
                Redir ->
                    C1 = C#sconf{redirect_map=RedirMap ++ [Redir]},
                    fload(FD, server_redirect, GC, C1, Lno+1, ?NEXTLINE)
            end;

        [Path, '=' | Rest] ->
            %% "Normalize" Path
            Path1 = filename:join([yaws_api:path_norm(Path)]),
            case parse_redirect(Path1, Rest, append, Lno) of
                {error, Str} ->
                    {error, Str};
                Redir ->
                    C1 = C#sconf{redirect_map=RedirMap ++ [Redir]},
                    fload(FD, server_redirect, GC, C1, Lno+1, ?NEXTLINE)
            end;

        ['<', "/redirect", '>'] ->
            fload(FD, server, GC, C, Lno+1, ?NEXTLINE);

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;

fload(FD, server_deflate, GC, C, Lno, Chars) ->
    Deflate = C#sconf.deflate_options,
    case toks(Lno, Chars) of
        [] ->
            fload(FD, server_deflate, GC, C, Lno+1, ?NEXTLINE);

        ["min_compress_size", '=', CSize] ->
            case (catch list_to_integer(CSize)) of
                I when is_integer(I), I > 0 ->
                    Deflate1 = Deflate#deflate{min_compress_size=I},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                _ when CSize == "nolimit" ->
                    Deflate1 = Deflate#deflate{min_compress_size=nolimit},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer > 0 at line ~w", [Lno])}
            end;

        ["mime_types", '=' | MimeTypes] ->
            case parse_compressible_mime_types(MimeTypes,
                                               Deflate#deflate.mime_types) of
                {ok, L} ->
                    Deflate1 = Deflate#deflate{mime_types=L},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                {error, Str} ->
                    {error, ?F("~s at line ~w", [Str, Lno])}
            end;

        ["compression_level", '=', CLevel] ->
            L = try
                    list_to_integer(CLevel)
                catch error:badarg ->
                        list_to_atom(CLevel)
                end,
            if
                L =:= none; L =:= default;
                L =:= best_compression; L =:= best_speed ->
                    Deflate1 = Deflate#deflate{compression_level=L},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                is_integer(L), L >= 0, L =< 9 ->
                    Deflate1 = Deflate#deflate{compression_level=L},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                true ->
                    {error, ?F("Bad compression level at line ~w", [Lno])}
            end;

        ["window_size", '=', WSize] ->
            case (catch list_to_integer(WSize)) of
                I when is_integer(I), I > 8, I < 16 ->
                    Deflate1 = Deflate#deflate{window_size=I * -1},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error,
                     ?F("Expect integer between 9..15 at line ~w",
                        [Lno])}
            end;

        ["mem_level", '=', MLevel] ->
            case (catch list_to_integer(MLevel)) of
                I when is_integer(I), I >= 1, I =< 9 ->
                    Deflate1 = Deflate#deflate{mem_level=I},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                _ ->
                    {error, ?F("Expect integer between 1..9 at line ~w", [Lno])}
            end;

        ["strategy", '=', Strategy] ->
            if
                Strategy =:= "default";
```

```erlang
                Strategy =:= "filtered";
                Strategy =:= "huffman_only" ->
                    Deflate1 = Deflate#deflate{strategy=list_to_atom(Strategy)},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                true ->
                    {error,
                     ?F("Unknown strategy ~p at line ~w", [Strategy, Lno])}
            end;

        ["use_gzip_static", '=', Bool] ->
            case is_bool(Bool) of
                {true, Val} ->
                    Deflate1 = Deflate#deflate{use_gzip_static=Val},
                    C1 = C#sconf{deflate_options=Deflate1},
                    fload(FD, server_deflate, GC, C1, Lno+1, ?NEXTLINE);
                false ->
                    {error, ?F("Expect true|false at line ~w", [Lno])}
            end;

        ['<', "/deflate", '>'] ->
            Deflate1 = case Deflate#deflate.mime_types of
                           [] ->
                               Deflate#deflate{
                                 mime_types = ?DEFAULT_COMPRESSIBLE_MIME_TYPES
                                };
                           _ ->
                               Deflate
                       end,
            C1 = C#sconf{deflate_options = Deflate1},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;

fload(FD, extra_cgi_vars, GC, C, Lno, Chars) ->
    [{Dir, Vars}|EVars] = C#sconf.extra_cgi_vars,
    case toks(Lno, Chars) of
        [] ->
            fload(FD, extra_cgi_vars, GC, C, Lno+1, ?NEXTLINE);

        [Var, '=', Val] ->
            C1 = C#sconf{extra_cgi_vars=[{Dir, [{Var, Val} | Vars]}|EVars]},
            fload(FD, extra_cgi_vars, GC, C1, Lno+1, ?NEXTLINE);

        ['<', "/extra_cgi_vars", '>'] ->
            C1 = C#sconf{extra_cgi_vars = [EVars | C#sconf.extra_cgi_vars]},
            fload(FD, server, GC, C1, Lno+1, ?NEXTLINE);

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;

fload(FD, rss, GC, C, Lno, Chars) ->
    case toks(Lno, Chars) of
        [] ->
            fload(FD, rss, GC, C, Lno+1, ?NEXTLINE);

        ["rss_id", '=', Value] ->   % mandatory !!
            put(rss_id, list_to_atom(Value)),
            fload(FD, rss, GC, C, Lno+1, ?NEXTLINE);

        ["rss_dir", '=', Value] ->   % mandatory !!
            put(rss, [{db_dir, Value} | get(rss)]),
            fload(FD, rss, GC, C, Lno+1, ?NEXTLINE);

        ["rss_expire", '=', Value] ->
            put(rss, [{expire, Value} | get(rss)]),
            fload(FD, rss, GC, C, Lno+1, ?NEXTLINE);

        ["rss_days", '=', Value] ->
            put(rss, [{days, Value} | get(rss)]),
            fload(FD, rss, GC, C, Lno+1, ?NEXTLINE);

        ["rss_rm_exp", '=', Value] ->
            put(rss, [{rm_exp, Value} | get(rss)]),
            fload(FD, rss, GC, C, Lno+1, ?NEXTLINE);

        ["rss_max", '=', Value] ->
            put(rss, [{rm_max, Value} | get(rss)]),
            fload(FD, rss, GC, C, Lno+1, ?NEXTLINE);

        ['<', "/rss", '>'] ->
            case get(rss_id) of
                undefined ->
                    {error, ?F("No rss_id specified at line ~w", [Lno])};
                RSSid ->
                    yaws_rss:open(RSSid, get(rss)),
                    fload(FD, server, GC, C, Lno+1, ?NEXTLINE)
            end;

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end;

fload(FD, opaque, GC, C, Lno, Chars) ->
    case toks(Lno, Chars) of
        [] ->
            fload(FD, opaque, GC, C, Lno+1, ?NEXTLINE);

        [Key, '=', Value] ->
            C1 = C#sconf{opaque = [{Key,Value} | C#sconf.opaque]},
            fload(FD, opaque, GC, C1, Lno+1, ?NEXTLINE);

        [Key, '='| Value] ->
            String_value = lists:flatten(
                                 lists:map(
                                   fun(Item) when is_atom(Item) ->
                                           atom_to_list(Item);
                                      (Item) ->
                                           Item
                                   end, Value)),
            C1 = C#sconf{opaque = [{Key, String_value} | C#sconf.opaque]},
            fload(FD, opaque, GC, C1, Lno+1, ?NEXTLINE);

        ['<', "/opaque", '>'] ->
            fload(FD, server, GC, C, Lno+1, ?NEXTLINE);

        [H|T] ->
            {error, ?F("Unexpected input ~p at line ~w", [[H|T], Lno])};
        Err ->
            Err
    end.

is_bool("true") ->
    {true, true};
is_bool("false") ->
    {true, false};
is_bool(_) ->
    false.


warn_dir(Type, Dir) ->
    case is_dir(Dir) of
        true ->
            true;
        false ->
            error_logger:format("Config Warning: Directory ~s "
                                "for ~s doesn't exist~n",
                                [Dir, Type]),
            false
    end.

is_dir(Val) ->
    case file:read_file_info(Val) of
        {ok, FI} when FI#file_info.type == directory ->
            true;
        _ ->
            false
    end.


is_file(Val) ->
    case file:read_file_info(Val) of
        {ok, FI} when FI#file_info.type == regular ->
            true;
        _ ->
            false
    end.

is_wildcard(Val) ->
    (lists:member($*, Val) orelse
     lists:member($?, Val) orelse
     (lists:member($[, Val) andalso lists:member($], Val)) orelse
     (lists:member(${, Val) andalso lists:member($}, Val))).


%% tokenizer
toks(Lno, Chars) ->
    toks(Lno, Chars, free, [], []). % two accumulators

toks(Lno, [$#|_T], Mode, Ack, Tack) ->
    toks(Lno, [], Mode, Ack, Tack);

toks(Lno, [H|T], free, Ack, Tack) ->
    %%?Debug("Char=~p", [H]),
    case {is_quote(H), is_string_char([H|T]),is_special(H), yaws:is_space(H)} of
        {_,_, _, true} ->
            toks(Lno, T, free, Ack, Tack);
        {_,_, true, _} ->
            toks(Lno, T, free, [], [list_to_atom([H]) | Tack]);
        {_,true, _,_} ->
            toks(Lno, T, string, [H], Tack);
        {_,utf8, _,_} ->
            toks(Lno, tl(T), string, [H, hd(T)], Tack);
        {true,_, _,_} ->
            toks(Lno, T, quote, [], Tack);
        {false, false, false, false} ->
            {error, ?F("Unexpected character  <~p / ~c> at line ~w",
                       [H,H, Lno])}
    end;
toks(Lno, [C|T], string, Ack, Tack) ->
    case {is_backquote(C), is_quote(C), is_string_char([C|T]), is_special(C),
          yaws:is_space(C)} of
        {true, _, _, _,_} ->
            toks(Lno, T, [backquote,string], Ack, Tack);
        {_, _, true, _,_} ->
            toks(Lno, T, string, [C|Ack], Tack);
        {_, _, utf8, _,_} ->
            toks(Lno, tl(T), string, [C, hd(T)|Ack], Tack);
        {_, _, _, true, _} ->
            toks(Lno, T, free, [], [list_to_atom([C]),lists:reverse(Ack)|Tack]);
        {_, true, _, _, _} ->
            toks(Lno, T, quote, [], [lists:reverse(Ack)|Tack]);
        {_, _, _, _, true} ->
            toks(Lno, T, free, [], [lists:reverse(Ack)|Tack]);
        {false, false, false, false, false} ->
            {error, ?F("Unexpected character  <~p / ~c> at line ~w",
                       [C, C, Lno])}
    end;
toks(Lno, [C|T], quote, Ack, Tack) ->
    case {is_quote(C), is_backquote(C)} of
        {true, _} ->
            toks(Lno, T, free, [], [lists:reverse(Ack)|Tack]);
        {_, true} ->
            toks(Lno, T, [backquote,quote], [C|Ack], Tack);
        {false, false} ->
            toks(Lno, T, quote, [C|Ack], Tack)
    end;
toks(Lno, [C|T], [backquote,Mode], Ack, Tack) ->
    toks(Lno, T, Mode, [C|Ack], Tack);
toks(_Lno, [], string, Ack, Tack) ->
    lists:reverse([lists:reverse(Ack) | Tack]);
toks(_Lno, [], free, _,Tack) ->
    lists:reverse(Tack).

is_quote(34) -> true ;  %% $" but emacs mode can't handle it
is_quote(_)  -> false.

is_backquote($\\) -> true ;
is_backquote(_)  -> false.

is_string_char([C|T]) ->
    if
        $a =< C, C =< $z ->
            true;
        $A =< C, C =< $Z ->
            true;
        $0 =< C, C =< $9 ->
            true;
        C == 195 , T /= [] ->
            %% FIXME check that [C, hd(T)] really is a char ?? how
            utf8;
        true ->
            lists:member(C, [$., $/, $:, $_, $-, $+, $~, $@, $*, $?])
    end.

is_special(C) ->
    lists:member(C, [$=, $[, $], ${, $}, $, ,$<, $>, $,]).

%% parse the argument string PLString which can either be the undefined
%% atom or a proplist. Currently the only supported keys are
%% fullsweep_after, min_heap_size, and min_bin_vheap_size. Any other
%% key/values are ignored.
parse_process_options(PLString) ->
    case erl_scan:string(PLString ++ ".") of
        {ok, PLTokens, _} ->
            case erl_parse:parse_term(PLTokens) of
                {ok, undefined} ->
                    {ok, []};
                {ok, []} ->
                    {ok, []};
                {ok, [Hd|_Tl]=PList} when is_atom(Hd); is_tuple(Hd) ->
                    %% create new safe proplist of desired options
                    {ok, proplists_int_copy([], PList, [fullsweep_after,
                                                        min_heap_size,
                                                        min_bin_vheap_size])}};
                _ ->
                    {error, "Expect undefined or proplist"}
            end;
        _ ->
            {error, "Expect undefined or proplist"}
    end.

%% copy proplist integer values for the given keys from the
%% Src proplist to the Dest proplist. Ignored keys that are not
%% found or have non-integer values. Returns the new Dest proplist.
proplists_int_copy(Dest, _Src, []) ->
    Dest;
proplists_int_copy(Dest, Src, [Key|NextKeys]) ->
    case proplists:get_value(Key, Src) of
        Val when is_integer(Val) ->
            proplists_int_copy([{Key, Val}|Dest], Src, NextKeys);
        _ ->
            proplists_int_copy(Dest, Src, NextKeys)
    end.

```

```erlang
parse_soap_srv_mods(['<', Module, ',' , Handler, ',', WsdlFile, '>' | Tail],
                    Ack) ->
    case is_file(WsdlFile) of
        true ->
            S = { {list_to_atom(Module), list_to_atom(Handler)}, WsdlFile},
            parse_soap_srv_mods(Tail, [S |Ack]);
        false ->
            {error, ?F("Bad wsdl file ~p", [WsdlFile])}
    end;

parse_soap_srv_mods(['<', Module, ',' , Handler, ',', WsdlFile, ',',
                     Prefix, '>' | Tail], Ack) ->
    case is_file(WsdlFile) of
        true ->
            S = { {list_to_atom(Module), list_to_atom(Handler)},
                  WsdlFile, Prefix},
            parse_soap_srv_mods(Tail, [S |Ack]);
        false ->
            {error, ?F("Bad wsdl file ~p", [WsdlFile])}
    end;

parse_soap_srv_mods([ SoapSrvMod | _Tail], _Ack) ->
    {error, ?F("Bad soap_srv_mods syntax: ~p", [SoapSrvMod])};

parse_soap_srv_mods([], Ack) ->
    {ok, Ack}.

parse_appmods(['<', PathElem, ',' , AppMod, '>' | Tail], Ack) ->
    S = {PathElem , list_to_atom(AppMod)},
    parse_appmods(Tail, [S |Ack]);

parse_appmods(['<', PathElem, ',' , AppMod, "exclude_paths" |Tail], Ack)->
    Paths = lists:takewhile(fun(X) -> X /= '>' end,
                            Tail),
    Tail2 = lists:dropwhile(fun(X) -> X /= '>' end,
                            Tail),
    Tail3 = tl(Tail2),

    S = {PathElem , list_to_atom(AppMod), lists:map(
                                            fun(Str) ->
                                                    string:tokens(Str, "/")
                                            end, Paths)},
    parse_appmods(Tail3, [S |Ack]);


parse_appmods([AppMod | Tail], Ack) ->
    %% just some simpleminded test to catch syntax errors in the config
    case AppMod of
        [Char] ->
            case is_special(Char) of
                true ->
                    {error, "Bad appmod syntax"};
                false ->
                    S = {AppMod, list_to_atom(AppMod)},
                    parse_appmods(Tail, [S | Ack])
            end;
        _ ->
            S = {AppMod, list_to_atom(AppMod)},
            parse_appmods(Tail, [S | Ack])
    end;

parse_appmods([], Ack) ->
    {ok, Ack}.


parse_revproxy([Prefix, Url]) ->
    parse_revproxy_url(Prefix, Url);
parse_revproxy([Prefix, Url, "intercept_mod", InterceptMod]) ->
    case parse_revproxy_url(Prefix, Url) of
        {ok, RP} ->
            {ok, RP#proxy_cfg{intercept_mod = list_to_atom(InterceptMod)}};
        Error ->
            Error
    end;
parse_revproxy([Prefix, Proto, '[', IPv6, ']', Rest, "intercept_mod", InterceptMod]) ->
    Url = Proto ++ "[" ++ IPv6 ++ "]" ++ Rest,
    parse_revproxy([Prefix, Url, "intercept_mod", InterceptMod]);
parse_revproxy([Prefix, Proto, '[', IPv6, ']', Rest]) ->
    Url = Proto ++ "[" ++ IPv6 ++ "]" ++ Rest,
    parse_revproxy([Prefix, Url]);
parse_revproxy(_Other) ->
    {error, syntax}.

parse_revproxy_url(Prefix, Url) ->
    case (catch yaws_api:parse_url(Url)) of
        {'EXIT', _} ->
            {error, url};
        URL when URL#url.path == "/" ->
            P = case lists:reverse(Prefix) of
                    [$/|_Tail] ->
                        Prefix;
                    Other ->
                        lists:reverse(Other)
                end,
            {ok, #proxy_cfg{prefix=P, url=URL}};
        _URL ->
            {error, "Can't revproxy to a URL with a path "}
    end.
```

```erlang


parse_expires(['<', MimeType, ',' , Expire, '>' | Tail], Acc) ->
    {EType, Value} =
        case string:tokens(Expire, "+") of
            ["always"] ->
                {always, 0};
            [Secs] ->
                {access, (catch list_to_integer(Secs))};
            ["access", Secs] ->
                {access, (catch list_to_integer(Secs))};
            ["modify", Secs] ->
                {modify, (catch list_to_integer(Secs))};
            _ ->
                {error, "Bad expires syntax"}
        end,
    if
        EType =:= error ->
            {EType, Value};
        not is_integer(Value) ->
            {error, "Bad expires syntax"};
        true ->
            case parse_mime_type(MimeType) of
                {ok, "*", "*"} ->
                    E = {all, EType, Value},
                    parse_expires(Tail, [E |Acc]);
                {ok, Type, "*"} ->
                    E = {{Type, all}, EType, Value},
                    parse_expires(Tail, [E |Acc]);
                {ok, _Type, _SubType} ->
                    E = {MimeType, EType, Value},
                    parse_expires(Tail, [E |Acc]);
                Error ->
                    Error
            end
    end;
parse_expires([], Acc)->
    {ok, Acc}.


parse_phpmod(['<', "cgi", ',', DefaultPhpPath, '>'], DefaultPhpPath) ->
    {ok, {cgi, DefaultPhpPath}};
parse_phpmod(['<', "cgi", ',', PhpPath, '>'], _) ->
    case is_file(PhpPath) of
        true ->
            {ok, {cgi, PhpPath}};
        false ->
            {error, ?F("~s is not a regular file", [PhpPath])}
    end;
parse_phpmod(['<', "fcgi", ',', HostPortSpec, '>'], _) ->
    case string_to_host_and_port(HostPortSpec) of
        {ok, Host, Port} ->
            {ok, {fcgi, {Host, Port}}};
        {error, Reason} ->
            {error, Reason}
    end;
parse_phpmod(['<', "fcgi", ',', '[', HostSpec, ']', PortSpec, '>'], _) ->
    case string_to_host_and_port("[" ++ HostSpec ++ "]" ++ PortSpec) of
        {ok, Host, Port} ->
            {ok, {fcgi, {Host, Port}}};
        {error, Reason} ->
            {error, Reason}
    end;
parse_phpmod(['<', "extern", ',', NodeModFunSpec, '>'], _) ->
    case string_to_node_mod_fun(NodeModFunSpec) of
        {ok, Node, Mod, Fun} ->
            {ok, {extern, {Node,Mod,Fun}}};
        {ok, Mod, Fun} ->
            {ok, {extern, {Mod,Fun}}};
        {error, Reason} ->
            {error, Reason}
    end.


parse_compressible_mime_types(_, all) ->
    {ok, all};
parse_compressible_mime_types(["all"|_], _Acc) ->
    {ok, all};
parse_compressible_mime_types(["defaults"|Rest], Acc) ->
    parse_compressible_mime_types(Rest, ?DEFAULT_COMPRESSIBLE_MIME_TYPES++Acc);
parse_compressible_mime_types([',' | Rest], Acc) ->
    parse_compressible_mime_types(Rest, Acc);
parse_compressible_mime_types([MimeType | Rest], Acc) ->
    case parse_mime_type(MimeType) of
        {ok, "*", "*"} ->
            {ok, all};
        {ok, Type, "*"} ->
            parse_compressible_mime_types(Rest, [{Type, all}|Acc]);
        {ok, Type, SubType} ->
            parse_compressible_mime_types(Rest, [{Type, SubType}|Acc]);
        Error ->
            Error
    end;
parse_compressible_mime_types([], Acc) ->
    {ok, Acc}.


parse_mime_type(MimeType) ->
```

```erlang
    Res = re:run(MimeType, "^([-\\w\+]+|\\*)/([-\\w\+\.]+|\\*)$",
                 [{capture, all_but_first, list}]),
    case Res of
        {match, [Type,SubType]} ->
            {ok, Type, SubType};
        nomatch ->
            {error, "Invalid MimeType"}
    end.


parse_index_files([]) ->
    ok;
parse_index_files([Idx|Rest]) ->
    case Idx of
        [$/|_] when Rest /= [] ->
            {error, "Only the last index should be absolute"};
        _ ->
            parse_index_files(Rest)
    end.

is_valid_mime_type(MimeType) ->
    case re:run(MimeType, "^[-\\w\+]+/[-\\w\+\.]+$", [{capture, none}]) of
        match   -> true;
        nomatch -> false
    end.

parse_mime_types(['<', MimeType, ',' | Tail], Acc0) ->
    Exts      = lists:takewhile(fun(X) -> X /= '>' end, Tail),
    [_|Tail2] = lists:dropwhile(fun(X) -> X /= '>' end, Tail),
    Acc1 = lists:foldl(fun(E, Acc) ->
                               lists:keystore(E, 1, Acc, {E, MimeType})
                       end, Acc0, Exts),
    case is_valid_mime_type(MimeType) of
        true  -> parse_mime_types(Tail2, Acc1);
        false -> {error, ?F("Invalid mime-type '~p'", [MimeType])}
    end;
parse_mime_types([], Acc)->
    {ok, lists:reverse(Acc)};
parse_mime_types(_, _) ->
    {error, "Unexpected tokens"}.

parse_charsets(['<', Charset, ',' | Tail], Acc0) ->
    Exts      = lists:takewhile(fun(X) -> X /= '>' end, Tail),
    [_|Tail2] = lists:dropwhile(fun(X) -> X /= '>' end, Tail),
    Acc1 = lists:foldl(fun(E, Acc) ->
                               lists:keystore(E, 1, Acc, {E, Charset})
                       end, Acc0, Exts),
    parse_charsets(Tail2, Acc1);
parse_charsets([], Acc)->
    {ok, lists:reverse(Acc)};
parse_charsets(_, _) ->
    {error, "Unexpected tokens"}.


parse_mime_types_info(Directive, Type, undefined, undefined) ->
    parse_mime_types_info(Directive, Type, #mime_types_info{});
parse_mime_types_info(Directive, Type, undefined, DefaultInfo) ->
    parse_mime_types_info(Directive, Type, DefaultInfo);
parse_mime_types_info(Directive, Type, Info, _) ->
    parse_mime_types_info(Directive, Type, Info).

parse_mime_types_info(default_type, Type, Info) ->
    case is_valid_mime_type(Type) of
        true  -> {ok, Info#mime_types_info{default_type=Type}};
        false -> {error, ?F("Invalid mime-type '~p'", [Type])}
    end;
parse_mime_types_info(default_charset, Charset, Info) ->
    {ok, Info#mime_types_info{default_charset=Charset}};
parse_mime_types_info(mime_types_file, File, Info) ->
    {ok, Info#mime_types_info{mime_types_file=File}};
parse_mime_types_info(add_types, NewTypes, Info) ->
    case parse_mime_types(NewTypes, Info#mime_types_info.types) of
        {ok, Types} -> {ok, Info#mime_types_info{types=Types}};
        Error       -> Error
    end;
parse_mime_types_info(add_charsets, NewCharsets, Info) ->
    case parse_charsets(NewCharsets, Info#mime_types_info.charsets) of
        {ok, Charsets} -> {ok, Info#mime_types_info{charsets=Charsets}};
        Error          -> Error
    end.


parse_nslookup_pref(Pref) ->
    parse_nslookup_pref(Pref, []).

parse_nslookup_pref(Empty, []) when Empty == [] orelse Empty == ['[', ']'] ->
    %% Get default value, if nslookup_pref = [].
    {ok, yaws:gconf_nslookup_pref(#gconf{})};
parse_nslookup_pref([C, Family | Rest], Result)
  when C == '[' orelse C == ',' ->
    case Family of
        "inet" ->
            case lists:member(inet, Result) of
                false -> parse_nslookup_pref(Rest, [inet | Result]);
                true  -> parse_nslookup_pref(Rest, Result)
            end;
        "inet6" ->
            case lists:member(inet6, Result) of
```

```erlang
                false -> parse_nslookup_pref(Rest, [inet6 | Result]);
                true  -> parse_nslookup_pref(Rest, Result)
            end;
        _ ->
            case Result of
                [PreviousFamily | _] ->
                    {error, ?F("Invalid nslookup_pref: invalid family or "
                        "token '~s', after family '~s'",
                        [Family, PreviousFamily])};
                [] ->
                    {error, ?F("Invalid nslookup_pref: invalid family or "
                        "token '~s'", [Family])}
            end
    end;
parse_nslookup_pref([']'], Result) ->
    {ok, lists:reverse(Result)};
parse_nslookup_pref([Invalid | _], []) ->
    {error, ?F("Invalid nslookup_pref: unexpected token '~s'", [Invalid])};
parse_nslookup_pref([Invalid | _], [Family | _]) ->
    {error, ?F("Invalid nslookup_pref: unexpected token '~s', "
        "after family '~s'", [Invalid, Family])}.


parse_redirect(Path, [Code, URL], Mode, Lno) ->
    case catch list_to_integer(Code) of
        I when is_integer(I), I >= 300, I =< 399 ->
            try yaws_api:parse_url(URL, sloppy) of
                U when is_record(U, url) ->
                    {Path, I, U, Mode}
            catch _:_ ->
                    {error, ?F("Bad redirect URL ~p at line ~w", [URL, Lno])}
            end;
        I when is_integer(I), I >= 100, I =< 599 ->
            %% Only relative path are authorized here
            try yaws_api:parse_url(URL, sloppy) of
                #url{scheme=undefined, host=[], port=undefined, path=P} ->
                    {Path, I, P, Mode};
                #url{} ->
                    {error, ?F("Bad redirect rule at line ~w: "
                               " Absolute URL is forbidden here", [Lno])}
            catch _:_ ->
                    {error, ?F("Bad redirect URL ~p at line ~w", [URL, Lno])}
            end;
        _ ->
            {error, ?F("Bad status code ~p at line ~w", [Code, Lno])}
    end;
parse_redirect(Path, [CodeOrUrl], Mode, Lno) ->
    case catch list_to_integer(CodeOrUrl) of
        I when is_integer(I), I >= 300, I =< 399 ->
            {error, ?F("Bad redirect rule at line ~w: "
                       "URL to redirect to is missing ", [Lno])};
        I when is_integer(I), I >= 100, I =< 599 ->
            {Path, I, undefined, Mode};
        I when is_integer(I) ->
            {error, ?F("Bad status code ~p at line ~w", [CodeOrUrl, Lno])};
        _ ->
            try yaws_api:parse_url(CodeOrUrl, sloppy) of
                #url{}=U ->
                    {Path, 302, U, Mode}
            catch _:_ ->
                    {error, ?F("Bad redirect URL ~p at line ~w",
                               [CodeOrUrl, Lno])}
            end
    end;
parse_redirect(_Path, _, _Mode, Lno) ->
    {error, ?F("Bad redirect rule at line ~w", [Lno])}.


ssl_start() ->
    case catch ssl:start() of
        ok ->
            ok;
        {error,{already_started,ssl}} ->
            ok;
        Err ->
            error_logger:format("Failed to start ssl: ~p~n", [Err])
    end.



%% search for an SC within Pairs that have the same, listen,port,ssl,severname
%% Return {Pid, SC, Scs} or false
%% Pairs is the pairs in yaws_server #state{}
search_sconf(GC, NewSC, Pairs) ->
    case lists:zf(
           fun({Pid, Scs = [SC|_]}) ->
                   case same_virt_srv(GC, NewSC, SC) of
                       true ->
                           case lists:keysearch(NewSC#sconf.servername,
                                                #sconf.servername, Scs) of
                               {value, Found} ->
                                   {true, {Pid, Found, Scs}};
                               false ->
                                   false
                           end;
                       false ->
                           false
                   end
```

```
          end, Pairs) of
        [] ->
            false;
        [{Pid, Found, Scs}] ->
            {Pid, Found, Scs};
        _Other ->
            error_logger:format("Fatal error, no two sconfs should "
                                " ever be considered equal ..",[]),
            erlang:error(fatal_conf)
    end.

%% find the group a new SC would belong to
search_group(GC, SC, Pairs) ->
    Fun =  fun({Pid, [S|Ss]}) ->
                   case same_virt_srv(GC, S, SC) of
                       true ->
                           {true, {Pid, [S|Ss]}};
                       false ->
                           false
                   end
           end,

    lists:zf(Fun, Pairs).


%% Return a new Pairs list with one SC updated
update_sconf(Gc, NewSc, Pos, Pairs) ->
    lists:map(
      fun({Pid, Scs}) ->
              case same_virt_srv(Gc, hd(Scs), NewSc) of
                  true ->
                      L2 = lists:keydelete(NewSc#sconf.servername,
                                           #sconf.servername, Scs),
                      {Pid, yaws:insert_at(NewSc, Pos, L2)};
                  false ->
                      {Pid, Scs}
              end
      end, Pairs).


%% return a new pairs list with SC removed
delete_sconf(Gc, OldSc, Pairs) ->
    lists:zf(
      fun({Pid, Scs}) ->
              case same_virt_srv(Gc, hd(Scs), OldSc) of
                  true ->
                      L2 = lists:keydelete(OldSc#sconf.servername,
                                           #sconf.servername, Scs),
                      {true, {Pid, L2}};
                  false ->
                      {true, {Pid, Scs}}
              end

      end, Pairs).



same_virt_srv(Gc, S, NewSc) when S#sconf.listen == NewSc#sconf.listen,
                                 S#sconf.port == NewSc#sconf.port ->
    if
        Gc#gconf.sni == disable orelse
        S#sconf.ssl == undefined orelse
        NewSc#sconf.ssl == undefined ->
            (S#sconf.ssl == NewSc#sconf.ssl);
        true ->
            true
    end;
same_virt_srv(_,_,_) ->
    false.


eq_sconfs(S1,S2) ->
    (S1#sconf.port == S2#sconf.port andalso
     S1#sconf.flags == S2#sconf.flags andalso
     S1#sconf.redirect_map == S2#sconf.redirect_map andalso
     S1#sconf.rhost == S2#sconf.rhost andalso
     S1#sconf.rmethod == S2#sconf.rmethod andalso
     S1#sconf.docroot == S2#sconf.docroot andalso
     S1#sconf.xtra_docroots == S2#sconf.xtra_docroots andalso
     S1#sconf.listen == S2#sconf.listen andalso
     S1#sconf.servername == S2#sconf.servername andalso
     S1#sconf.yaws == S2#sconf.yaws andalso
     S1#sconf.ssl == S2#sconf.ssl andalso
     S1#sconf.authdirs == S2#sconf.authdirs andalso
     S1#sconf.partial_post_size == S2#sconf.partial_post_size andalso
     S1#sconf.appmods == S2#sconf.appmods andalso
     S1#sconf.expires == S2#sconf.expires andalso
     S1#sconf.errormod_401 == S2#sconf.errormod_401 andalso
     S1#sconf.errormod_404 == S2#sconf.errormod_404 andalso
     S1#sconf.errormod_crash == S2#sconf.errormod_crash andalso
     S1#sconf.arg_rewrite_mod == S2#sconf.arg_rewrite_mod andalso
     S1#sconf.logger_mod == S2#sconf.logger_mod andalso
     S1#sconf.opaque == S2#sconf.opaque andalso
     S1#sconf.start_mod == S2#sconf.start_mod andalso
     S1#sconf.allowed_scripts == S2#sconf.allowed_scripts andalso
     S1#sconf.tilde_allowed_scripts == S2#sconf.tilde_allowed_scripts andalso
     S1#sconf.index_files == S2#sconf.index_files andalso
     S1#sconf.revproxy == S2#sconf.revproxy andalso
```

```erlang
     S1#sconf.soptions == S2#sconf.soptions andalso
     S1#sconf.extra_cgi_vars == S2#sconf.extra_cgi_vars andalso
     S1#sconf.stats == S2#sconf.stats andalso
     S1#sconf.fcgi_app_server == S2#sconf.fcgi_app_server andalso
     S1#sconf.php_handler == S2#sconf.php_handler andalso
     S1#sconf.shaper == S2#sconf.shaper andalso
     S1#sconf.deflate_options == S2#sconf.deflate_options andalso
     S1#sconf.mime_types_info == S2#sconf.mime_types_info andalso
     S1#sconf.dispatch_mod == S2#sconf.dispatch_mod andalso
     S1#sconf.extra_response_headers == S2#sconf.extra_response_headers).

%% This is the version of setconf that performs a
%% soft reconfig, it requires the args to be checked.
soft_setconf(GC, Groups, OLDGC, OldGroups) ->
    if
        GC /= OLDGC ->
            yaws_trace:setup(GC),
            update_gconf(GC);
        true ->
            ok
    end,
    compile_and_load_src_dir(GC),
    Grps = load_mime_types_module(GC, Groups),
    Rems = remove_old_scs(GC, lists:flatten(OldGroups), Grps),
    Adds = soft_setconf_scs(GC, lists:flatten(Grps), 1, OldGroups),
    lists:foreach(
      fun({delete_sconf, SC}) ->
              delete_sconf(SC);
         ({add_sconf, N, SC}) ->
              add_sconf(N, SC);
         ({update_sconf, N, SC}) ->
              update_sconf(N, SC)
      end, Rems ++ Adds).



hard_setconf(GC, Groups) ->
    gen_server:call(yaws_server,{setconf, GC, Groups}, infinity).


remove_old_scs(Gc, [Sc|Scs], NewGroups) ->
    case find_group(Gc, Sc, NewGroups) of
        false ->
            [{delete_sconf, Sc} |remove_old_scs(Gc, Scs, NewGroups)];
        {true, G} ->
            case find_sc(Sc, G) of
                false ->
                    [{delete_sconf, Sc} | remove_old_scs(Gc, Scs, NewGroups)];
                _ ->
                    remove_old_scs(Gc, Scs, NewGroups)
            end
    end;
remove_old_scs(_, [],_) ->
    [].

soft_setconf_scs(Gc, [Sc|Scs], N, OldGroups) ->
    case find_group(Gc, Sc, OldGroups) of
        false ->
            [{add_sconf,N,Sc} | soft_setconf_scs(Gc, Scs, N+1, OldGroups)];
        {true, G} ->
            case find_sc(Sc, G) of
                false ->
                    [{add_sconf,N,Sc} | soft_setconf_scs(Gc, Scs,N+1,OldGroups)];
                {true, _OldSc} ->
                    [{update_sconf,N,Sc} | soft_setconf_scs(Gc, Scs,N+1,OldGroups)]
            end
    end;
soft_setconf_scs(_,[], _, _) ->
    [].


%% checking code

can_hard_gc(New, Old) ->
    if
        Old == undefined ->
            true;
        New#gconf.yaws_dir == Old#gconf.yaws_dir,
        New#gconf.runmods == Old#gconf.runmods,
        New#gconf.logdir == Old#gconf.logdir ->
            true;
        true ->
            false
    end.



can_soft_setconf(NEWGC, NewGroups, OLDGC, OldGroups) ->
    can_soft_gc(NEWGC, OLDGC) andalso
        can_soft_sconf(NEWGC, lists:flatten(NewGroups), OldGroups).

can_soft_gc(G1, G2) ->
    if
        G1#gconf.flags == G2#gconf.flags,
        G1#gconf.logdir == G2#gconf.logdir,
        G1#gconf.log_wrap_size == G2#gconf.log_wrap_size,
        G1#gconf.sni == G2#gconf.sni,
        G1#gconf.id == G2#gconf.id ->
```

```
            true;
        true ->
            false
    end.


can_soft_sconf(Gc, [Sc|Scs], OldGroups) ->
    case find_group(Gc, Sc, OldGroups) of
        false ->
            can_soft_sconf(Gc, Scs, OldGroups);
        {true, G} ->
            case find_sc(Sc, G) of
                false ->
                    can_soft_sconf(Gc, Scs, OldGroups);
                {true, Old} when Old#sconf.start_mod /= Sc#sconf.start_mod ->
                    false;
                {true, Old} ->
                    case
                        {proplists:get_value(listen_opts, Old#sconf.soptions),
                         proplists:get_value(listen_opts, Sc#sconf.soptions)} of
                        {Opts, Opts} ->
                            can_soft_sconf(Gc, Scs, OldGroups);
                        _ ->
                            false
                    end
            end
    end;
can_soft_sconf(_, [], _) ->
    true.


find_group(GC, SC, [G|Gs]) ->
    case same_virt_srv(GC, SC, hd(G)) of
        true ->
            {true, G};
        false ->
            find_group(GC, SC, Gs)
    end;
find_group(_,_,[]) ->
    false.

find_sc(SC, [S|Ss]) ->
    if SC#sconf.servername  == S#sconf.servername  ->
            {true, S};
       true ->
            find_sc(SC, Ss)
    end;
find_sc(_SC,[]) ->
    false.


verify_upgrade_args(GC, Groups0) when is_record(GC, gconf) ->
    SCs0 = lists:flatten(Groups0),
    case lists:all(fun(SC) -> is_record(SC, sconf) end, SCs0) of
        true ->
            %% Embedded code may give appmods as a list of strings, or
            %% appmods can be {StringPathElem,ModAtom} or
            %% {StringPathElem,ModAtom,ExcludePathsList} tuples. Handle
            %% all possible variants here.
            SCs1 = lists:map(
                     fun(SC) ->
                             SC#sconf{appmods =
                                          lists:map(
                                            fun({PE, Mod}) ->
                                                    {PE, Mod};
                                               ({PE,Mod,Ex}) ->
                                                    {PE,Mod,Ex};
                                               (AM) when is_list(AM) ->
                                                    {AM,list_to_atom(AM)};
                                               (AM) when is_atom(AM) ->
                                                    {atom_to_list(AM), AM}
                                            end,
                                            SC#sconf.appmods)}
                     end, SCs0),
            case catch validate_cs(GC, SCs1) of
                {ok, GC, Groups1} -> {GC, Groups1};
                {error, Reason}   -> erlang:error(Reason);
                _                 -> erlang:error(badgroups)
            end;
        false ->
            erlang:error(badgroups)
    end.



add_sconf(SC) ->
    add_sconf(-1, SC).

add_sconf(Pos, SC0) ->
    {ok, SC1} = gen_server:call(yaws_server, {add_sconf, Pos, SC0}, infinity),
    ok = yaws_log:add_sconf(SC1),
    {ok, SC1}.

update_sconf(Pos, SC) ->
    gen_server:call(yaws_server, {update_sconf, Pos, SC}, infinity).

delete_sconf(SC) ->
    ok = gen_server:call(yaws_server, {delete_sconf, SC}, infinity),
    ok = yaws_log:del_sconf(SC).

update_gconf(GC) ->
    ok = gen_server:call(yaws_server, {update_gconf, GC}, infinity).


parse_auth_ips([], Result) ->
    Result;
parse_auth_ips([Str|Rest], Result) ->
    try
        parse_auth_ips(Rest, [yaws:parse_ipmask(Str)|Result])
    catch
        _:_ -> parse_auth_ips(Rest, Result)
    end.

parse_auth_user(User, Lno) ->
    try
        [Name, Passwd] = string:tokens(User, ":"),
        case re:run(Passwd, "{([^}]+)}(?:\\$([^$]+)\\$)?(.+)", [{capture,all_but_first,list}]) of
            {match, [Algo, B64Salt, B64Hash]} ->
                case parse_auth_user(Name, Algo, B64Salt, B64Hash) of
                    {ok, Res} ->
                        Res;
                    {error, bad_algo} ->
                        {error, ?F("Unsupported hash algorithm '~p' at line ~w",
                                   [Algo, Lno])};
                    {error, bad_user} ->
                        {error, ?F("Invalid user at line ~w", [Lno])}
                end;
            _ ->
                Salt = crypto:strong_rand_bytes(32),
                {Name, sha256, Salt, crypto:hash(sha256, [Salt, Passwd])}
        end
    catch
        _:_ ->
            {error, ?F("Invalid user at line ~w", [Lno])}
    end.

parse_auth_user(User, Algo, B64Salt, B64Hash) ->
    try
        if
            Algo == "md5"    orelse Algo == "sha"    orelse
            Algo == "sha224" orelse Algo == "sha256" orelse
            Algo == "sha384" orelse Algo == "sha512" orelse
            Algo == "ripemd160" ->
                Salt = base64:decode(B64Salt),
                Hash = base64:decode(B64Hash),
                {ok, {User, list_to_atom(Algo), Salt, Hash}};
            true ->
                {error, bad_algo}
        end
    catch
        _:_ -> {error, bad_user}
    end.


subconfigfiles(FD, Name, Lno) ->
    {ok, Config} = file:pid2name(FD),
    ConfPath = filename:dirname(filename:absname(Config)),
    File = filename:absname(Name, ConfPath),
    case {is_file(File), is_wildcard(Name)} of
        {true,_} ->
            {ok, [File]};
        {false,true} ->
            Names = filelib:wildcard(Name, ConfPath),
            Files = [filename:absname(N, ConfPath) || N <- lists:sort(Names)],
            {ok, lists:filter(fun filter_subconfigfile/1, Files)};
        {false,false} ->
            {error, ?F("Expect filename or wildcard at line ~w"
                       " (subconfig: ~s)", [Lno, Name])}
    end.

subconfigdir(FD, Name, Lno) ->
    {ok, Config} = file:pid2name(FD),
    ConfPath = filename:dirname(filename:absname(Config)),
    Dir = filename:absname(Name, ConfPath),
    case is_dir(Dir) of
        true ->
            case file:list_dir(Dir) of
                {ok, Names} ->
                    Files = [filename:absname(N, Dir) || N <- lists:sort(Names)],
                    {ok, lists:filter(fun filter_subconfigfile/1, Files)};
                {error, Error} ->
                    {error, ?F("Directory ~s is not readable: ~s",
                               [Name, Error])}
            end;
        false ->
            {error, ?F("Expect directory at line ~w (subconfdir: ~s)",
                       [Lno, Dir])}
    end.

filter_subconfigfile(File) ->
    case filename:basename(File) of
        [$.|_] ->
            error_logger:info_msg("Yaws: Ignore subconfig file ~s~n", [File]),
            false;
        _ ->
            true
```

```erlang
    end.

fload_subconfigfiles([], global, GC, Cs) ->
    {ok, GC, Cs};
fload_subconfigfiles([File|Files], global, GC, Cs) ->
    error_logger:info_msg("Yaws: Using global subconfig file ~s~n", [File]),
    case file:open(File, [read]) of
        {ok, FD} ->
            R = (catch fload(FD, GC, Cs, 1, ?NEXTLINE)),
            ?Debug("FLOAD(~s): ~p", [File, R]),
            case R of
                {ok, GC1, Cs1} -> fload_subconfigfiles(Files, global, GC1, Cs1);
                Err            -> Err
            end;
        Err ->
            {error, ?F("Can't open subconfig file ~s: ~p", [File,Err])}
    end;
fload_subconfigfiles([], server, GC, C) ->
    {ok, GC, C};
fload_subconfigfiles([File|Files], server, GC, C) ->
    error_logger:info_msg("Yaws: Using server subconfig file ~s~n", [File]),
    case file:open(File, [read]) of
        {ok, FD} ->
            R = (catch fload(FD, server, GC, C, 1, ?NEXTLINE)),
            ?Debug("FLOAD(~s): ~p", [File, R]),
            case R of
                {ok, GC1, C1, _, eof} ->
                    fload_subconfigfiles(Files, server, GC1, C1);
                {ok, _, _, Lno, ['<', "/server", '>']} ->
                    {error, ?F("Unexpected closing tag in subconfgile ~s"
                               " at line ~w ", [File, Lno])};
                Err ->
                    Err
            end;
        Err ->
            {error, ?F("Can't open subconfig file ~s: ~p", [File,Err])}
    end.


str2term(Str0) ->
    Str=Str0++".",
    {ok,Tokens,_EndLine} = erl_scan:string(Str),
    {ok,AbsForm} = erl_parse:parse_exprs(Tokens),
    {value,Value,_Bs} = erl_eval:exprs(AbsForm, erl_eval:new_bindings()),
    Value.

check_ciphers([], _) ->
    ok;
check_ciphers([Spec|Specs], L) ->
    case lists:member(Spec, L) of
        true ->
            check_ciphers(Specs, L);
        false ->
            {error, ?F("Bad cipherspec ~p",[Spec])}
    end;
check_ciphers(X,_) ->
    {error, ?F("Bad cipherspec ~p",[X])}.

check_eccs(From_conf, Available) ->
    case From_conf -- Available of
        [] -> ok;
        Bad -> {error, ?F("Bad elliptic curves ~p",[Bad])}
    end.

io_get_line(FD, Prompt, Acc) ->
    Next = io:get_line(FD, Prompt),
    if
        is_list(Next) ->
            case lists:reverse(Next) of
                [$\n, $\\ |More] ->
                    io_get_line(FD, Prompt, Acc ++ lists:reverse(More));
                _ ->
                    Acc ++ Next
            end;
        true ->
            Next
    end.

update_soptions(SC, Name, Key, Value) ->
    Opts0 = proplists:get_value(Name, SC#sconf.soptions),
    Opts1 = lists:keystore(Key, 1, Opts0, {Key, Value}),
    SOpts = lists:keystore(Name, 1, SC#sconf.soptions, {Name, Opts1}),
    SC#sconf{soptions = SOpts}.

set_sendfile_flags(GC, "erlang") ->
    {ok, ?gc_set_use_erlang_sendfile(GC, true)};
set_sendfile_flags(GC, "disable") ->
    {ok, ?gc_set_use_erlang_sendfile(GC, false)};
set_sendfile_flags(_, _) ->
    {error, "Expect erlang|disable"}.
```