#!KAMAILIO

#

# This config file implements the basic P-CSCF functionality

# - web: http://www.kamailio.org

# - git: http://sip-router.org

#

# Refer to the Core CookBook at http://www.kamailio.org/dokuwiki/doku.php

# for an explanation of possible statements, functions and parameters.

#

# Direct your questions about this file to: <sr-users@lists.sip-router.org>.

#

# For more information about the various parameters, functions and statements

# try http://sip-router.org/wiki/ .

#

####### Defined Values #########

# \*\*\* Value defines - IDs used later in config

# - flags

# FLT\_ - per transaction (message) flags

# FLB\_ - per branch flags

#!define FLT\_CAPTURE 1

#!define FLT\_DIALOG 2

#!define DLG\_TIMEOUT\_AVP "i:1"

#!define RR\_CUSTOM\_USER\_AVP "i:2"

##!define DISPATCHER\_DST\_AVP "i:3"

##!define DISPATCHER\_GRP\_AVP "i:4"

##!define DISPATCHER\_CNT\_AVP "i:5"

#!define PDB\_CARRIER "i:6"

####### Global Parameters #########

include\_file "scscf.cfg"

#!ifdef WITH\_DEBUG

debug=5

log\_stderror=no

sip\_warning=yes

#!else

debug=2

log\_stderror=no

sip\_warning=no

#!endif

### for foreground operation by FE

fork=no

log\_stderror=yes

children=4

alias=HOSTNAME

user\_agent\_header="User-Agent: Kamailio S-CSCF"

server\_header="Server: Kamailio S-CSCF"

/\* comment the next line to enable the auto discovery of local aliases

based on reverse DNS on IPs (default on) \*/

auto\_aliases=no

check\_via=no # (cmd. line: -v)

dns=no # (cmd. line: -r)

rev\_dns=no # (cmd. line: -R)

# Do SRV-Loadbalancing:

#dns\_srv\_lb=yes

# Always: Also try IPv6:

#dns\_try\_ipv6=yes

# Always prefer IPv6:

dns\_cache\_flags=6

# DNS-Based failover

use\_dns\_failover = off

# Query NAPTR-Records as well:

dns\_try\_naptr=no

#!ifdef WITH\_XMLRPC

#!ifndef WITH\_TCP

#!define WITH\_TCP

#!endif

#!ifndef TCP\_PROCESSES

# Number of TCP Processes

#!define TCP\_PROCESSES 3

#!endif

#!endif

#!ifdef WITH\_TCP

# life time of TCP connection when there is no traffic

# - a bit higher than registration expires to cope with UA behind NAT

tcp\_connection\_lifetime=3615

#!ifdef TCP\_PROCESSES

tcp\_children=TCP\_PROCESSES

#!endif

#!else

disable\_tcp=yes

#!endif

#children=64

system.shutdownmode = 0 desc "System shutdown mode"

# ------------------ module loading ----------------------------------

#mpath="/usr/lib64/kamailio/modules\_k/:/usr/lib64/kamailio/modules/:/usr/lib/kamailio/modules\_k/:/usr/lib/kamailio/modules/"

# (we try both the lib64 and the lib directory)

mpath="/usr/local/lib64/kamailio/modules/"

loadmodule "tm.so"

loadmodule "pv.so"

loadmodule "sl.so"

loadmodule "rr.so"

loadmodule "dialog\_ng.so"

loadmodule "textops.so"

loadmodule "maxfwd.so"

loadmodule "xlog.so"

loadmodule "sanity.so"

loadmodule "siputils.so"

loadmodule "kex.so"

#!ifdef DB\_URL

loadmodule "presence"

#!endif

#!ifdef DB\_URL

loadmodule "db\_mysql"

#!ifdef DB\_URL2

loadmodule "db\_cluster"

#!endif

#!endif

#loadmodule "dispatcher"

loadmodule "enum"

loadmodule "uac"

# Control interfaces:

#loadmodule "ctl"

#loadmodule "cfg\_rpc"

#loadmodule "mi\_rpc"

#loadmodule "mi\_fifo"

#!ifdef WITH\_XMLRPC

loadmodule "xmlrpc"

#!endif

loadmodule "cdp.so"

loadmodule "cdp\_avp.so"

loadmodule "ims\_usrloc\_scscf.so"

loadmodule "ims\_registrar\_scscf.so"

loadmodule "ims\_auth.so"

loadmodule "ims\_isc.so"

#!ifdef WITH\_RO

loadmodule "ims\_charging.so"

#!endif

#!ifdef CAPTURE\_NODE

loadmodule "siptrace.so"

#!endif

#!ifdef WITH\_DEBUG

loadmodule "debugger.so"

#!endif

# ----------------- setting module-specific parameters ---------------

#!ifdef DB\_URL2

# ----- db\_cluster params -----

modparam("db\_cluster", "connection", DB\_URL)

modparam("db\_cluster", "connection", DB\_URL2)

modparam("db\_cluster", "cluster", "cluster1=>con1=2s2s;con2=1s1s")

#!endif

# ----- presence params -----

#!ifdef DB\_URL

#!ifdef DB\_URL2

modparam("presence", "db\_url", "cluster://cluster1")

#!else

modparam("presence", "db\_url", DB\_URL)

#!endif

#modparam("presence", "fallback2db", 1)

modparam("presence", "db\_update\_period", 20)

#!endif

# ----- mi\_fifo params -----

#modparam("mi\_fifo", "fifo\_name", "/tmp/kamailio\_fifo")

#modparam("mi\_fifo", "fifo\_mode", 0666)

#modparam("mi\_fifo", "fifo\_user", "kamailio")

#modparam("mi\_fifo", "fifo\_group", "kamailio")

# ----- tm params -----

# auto-discard branches from previous serial forking leg

modparam("tm", "failure\_reply\_mode", 3)

# default retransmission timeout: 10sec

modparam("tm", "fr\_timer", 10000)

# default invite retransmission timeout after 1xx: 120sec

modparam("tm", "fr\_inv\_timer", 120000)

# Don't reply automatically with "100 Trying"

modparam("tm", "auto\_inv\_100", 0)

# ----- rr params -----

# add value to ;lr param to cope with most of the UAs

modparam("rr", "enable\_full\_lr", 1)

# append from tag to the RR

modparam("rr", "append\_fromtag", 1)

# add a Username to RR-Header

modparam("rr", "add\_username", 1)

# Take User from a custom AVP

modparam("rr", "custom\_user\_avp", "$avp(RR\_CUSTOM\_USER\_AVP)")

# -- usrloc params --

modparam("ims\_usrloc\_scscf", "enable\_debug\_file", 0)

modparam("ims\_usrloc\_scscf", "matching\_mode", 0)

modparam("ims\_registrar\_scscf", "max\_contacts", 5);

#!ifdef DB\_URL

#!ifdef DB\_URL2

modparam("ims\_usrloc\_scscf", "db\_url", "cluster://cluster1")

#!else

modparam("ims\_usrloc\_scscf", "db\_url", DB\_URL)

#!endif

modparam("ims\_usrloc\_scscf", "db\_mode", 1)

#!endif

modparam("ims\_registrar\_scscf", "default\_expires", 604800)

modparam("ims\_registrar\_scscf", "min\_expires", 3600)

modparam("ims\_registrar\_scscf", "max\_expires", 604800)

modparam("ims\_registrar\_scscf", "subscription\_default\_expires", 654800)

modparam("ims\_registrar\_scscf", "subscription\_min\_expires", 3700)

modparam("ims\_registrar\_scscf", "subscription\_max\_expires", 605800)

# -- CDP params --

modparam("cdp","config\_file","/usr/local/etc/kamailio/scscf.xml")

# -- dialog\_ng params --

modparam("dialog\_ng", "dlg\_flag", FLT\_DIALOG)

modparam("dialog\_ng", "timeout\_avp", "$avp(DLG\_TIMEOUT\_AVP)")

modparam("dialog\_ng", "detect\_spirals", 1)

modparam("dialog\_ng", "profiles\_no\_value", "orig ; term")

#!ifdef WITH\_XMLRPC

# ----- xmlrpc params -----

modparam("xmlrpc", "route", "XMLRPC");

modparam("xmlrpc", "url\_match", "^/RPC")

#!endif

#!ifdef WITH\_DEBUG

# ----- debugger params -----

modparam("debugger", "cfgtrace", 1)

#!endif

#!ifdef CAPTURE\_NODE

# Destination, where to send the traffic

modparam("siptrace", "duplicate\_uri", CAPTURE\_NODE)

# Trace all traffic

modparam("siptrace", "trace\_on", 1)

modparam("siptrace", "trace\_to\_database", 0)

modparam("siptrace", "trace\_flag", FLT\_CAPTURE)

modparam("siptrace", "hep\_mode\_on", 1)

#!endif

# -- ims\_auth params --

modparam("ims\_auth", "name", URI)

modparam("ims\_auth", "registration\_default\_algorithm", REG\_AUTH\_DEFAULT\_ALG)

#!ifdef CXDX\_FORCED\_PEER

modparam("ims\_auth", "cxdx\_forced\_peer", CXDX\_FORCED\_PEER)

#!endif

modparam("ims\_auth", "cxdx\_dest\_realm", NETWORKNAME)

modparam("ims\_auth", "av\_check\_only\_impu", 1)

# -- ims\_registrar\_scscf params --

#!ifdef WITH\_DEBUG

modparam("ims\_registrar\_scscf", "default\_expires", 60)

modparam("ims\_registrar\_scscf", "min\_expires", 60)

modparam("ims\_registrar\_scscf", "max\_expires", 60)

#!else

modparam("ims\_registrar\_scscf", "default\_expires", 600)

modparam("ims\_registrar\_scscf", "min\_expires", 300)

modparam("ims\_registrar\_scscf", "max\_expires", 3600)

#!endif

modparam("ims\_registrar\_scscf", "use\_path", 1)

modparam("ims\_registrar\_scscf", "support\_wildcardPSI",1)

modparam("ims\_registrar\_scscf", "user\_data\_xsd","/usr/local/etc/kamailio/CxDataType\_Rel7.xsd")

modparam("ims\_registrar\_scscf", "scscf\_name", URI)

modparam("ims\_registrar\_scscf", "cxdx\_dest\_realm", NETWORKNAME)

# ----- ims\_isc params -----

modparam("ims\_isc", "my\_uri", HOSTNAME)

#!ifdef WITH\_RO

# ----- ims\_diameter\_ro params -----

modparam("ims\_charging", "origin\_host", HOSTNAME);

modparam("ims\_charging", "origin\_realm", NETWORKNAME);

#!ifdef RO\_FORCED\_PEER

modparam("ims\_charging", "ro\_forced\_peer", RO\_FORCED\_PEER);

#!endif

modparam("ims\_charging", "destination\_host", RO\_DESTINATION);

modparam("ims\_charging", "destination\_realm", NETWORKNAME);

modparam("ims\_charging","service\_context\_id\_root", RO\_ROOT);

modparam("ims\_charging","service\_context\_id\_ext", RO\_EXT);

modparam("ims\_charging","service\_context\_id\_mnc", RO\_MNC);

modparam("ims\_charging","service\_context\_id\_mcc", RO\_MCC);

modparam("ims\_charging","service\_context\_id\_release", RO\_RELEASE);

modparam("ims\_charging","interim\_update\_credits",30);

modparam("ims\_charging","timer\_buffer",5);

#!endif

# ----- enum params -----

modparam("enum", "domain\_suffix", ENUM\_SUFFIX)

# ----- sanity params -----

modparam("sanity", "autodrop", 0)

# -- presence params --

modparam("presence", "subs\_remove\_match", 1) #means we match subscriptions on all attributes

# ----------------- Settings for Dispatcher ---------------

#modparam("dispatcher", "list\_file", "usr/local/etc/kamailio/dispatcher.list")

# Dispatcher: Enable Failover-Support

#modparam("dispatcher", "flags", 2)

# Dispatcher: Overwrite Destination address, if required.

#modparam("dispatcher", "force\_dst", 1)

# AVP's required for Fail-Over-Support:

#modparam("dispatcher", "dst\_avp", "$avp(DISPATCHER\_DST\_AVP)")

#modparam("dispatcher", "grp\_avp", "$avp(DISPATCHER\_GRP\_AVP)")

#modparam("dispatcher", "cnt\_avp", "$avp(DISPATCHER\_CNT\_AVP)")

# Try to recover disabled destinations every 15 seconds.

#modparam("dispatcher", "ds\_ping\_interval", 15)

# Actively query the gateways:

#modparam("dispatcher", "ds\_probing\_mode", 1)

####### Routing Logic ########

# Main SIP request routing logic

# - processing of any incoming SIP request starts with this route

route {

#trace every request

xlog("L\_INFO", "$C(xb) [$rm] from [$fu] to [$tu] $C(xx) \n");

#!ifdef WITH\_DEBUG

xlog("L\_ERR", "$rm ($fu ($si:$sp) to $tu, $ci)\n");

#!endif

# per request initial checks

route(REQINIT);

# Handle Registrations:

if (is\_method("REGISTER")) {

route(REGISTER);

exit;

}

# we need to support subscription to reg event

if (is\_method("SUBSCRIBE") && search("^(Event|o)([ \t]\*):([ \t]\*)reg")) {

route(SUBSCRIBE);

break;

}

if (is\_method("PUBLISH") && search("^(Event|o)([ \t]\*):([ \t]\*)reg")) {

route(PUBLISH);

break;

}

#Set DLG flag to track dialogs using dialog2

if (!is\_method("REGISTER|SUBSCRIBE"))

setflag(FLT\_DIALOG);

# Evaluate Route-Header and set $route\_uri

loose\_route();

if (is\_method("CANCEL|ACK")) {

t\_relay();

exit;

}

if (($route\_uri =~ "sip:orig@"+HOSTNAME\_ESC+".\*") || isc\_from\_as("orig")) {

# we need something like this to assign SCSCF to unregistered user for services

# support for AS origination on behalf of unregistered useri

# can use the registrar is\_registered methods - must see if we need to check orig or term?

# Originating

route(orig);

break;

} else {

isc\_from\_as("term");

if ($retcode == -2) {

# Treat as originating, since it was retargeted:

route(orig);

break;

}

if ((is\_in\_profile("orig") || has\_totag()) && ($route\_uri =~ "sip:mo@"+".\*")) {

route(orig\_subsequent);

break;

}

if ((is\_in\_profile("term") || has\_totag()) && ($route\_uri =~ "sip:mt@"+".\*")) {

route(term\_subsequent);

break;

}

# Terminating

if (uri=~"sip:(.\*)@"+NETWORKNAME\_ESC +"(.\*)" || uri=~"tel:.\*") {

if (!term\_impu\_registered("location")) {

xlog("L\_DBG", "We need to do an UNREG server SAR assignemnt");

assign\_server\_unreg("UNREG\_SAR\_REPLY", "location", "term");

exit;

}

} else {

sl\_send\_reply("403","Forbidden - Dialog not found on S-CSCF or Terminating user not suitable for unregistered services");

exit();

}

route(term);

break;

}

}

route[UNREG\_SAR\_REPLY]

{

xlog("L\_DBG","saa\_return code is $avp(s:saa\_return\_code)\n");

switch ($avp(s:saa\_return\_code)){

case 1: #success

xlog("L\_DBG", "SAR success - will route message\n");

route(term);

break;

case -1: #failure

xlog("L\_ERR", "SAR failure - error response sent from module\n");

break;

case -2: #error

xlog("L\_ERR", "SAR error - error response sent from module\n");

break;

default:

xlog("L\_ERR", "Unknown return code from SAR, value is [$avp(s:saa\_return\_code)]\n");

break;

}

exit;

}

######################################################################

# Helper routes (Basic-Checks, NAT-Handling/RTP-Control, XML-RPC)

######################################################################

# Per SIP request initial checks

route[REQINIT] {

# Trace this message

#!ifdef CAPTURE\_NODE

sip\_trace();

setflag(FLT\_CAPTURE);

#!endif

if (!mf\_process\_maxfwd\_header("10")) {

sl\_send\_reply("483","Too Many Hops");

exit;

}

if(!sanity\_check("1511", "7")) {

xlog("Malformed SIP message from $si:$sp\n");

exit;

}

# Check for shutdown mode:

if (!has\_totag() && ($sel(cfg\_get.system.shutdownmode) > 0)) {

send\_reply("503", "Server shutting down");

exit;

}

# Reply to OPTIONS:

if (is\_method("OPTIONS") && (uri==myself)) {

options\_reply();

exit;

}

# Ignore Re-Transmits:

if (t\_lookup\_request()) {

exit;

}

if (is\_method("INVITE")) {

send\_reply("100", "Trying");

}

}

######################################################################

# Publish route

######################################################################

route[PUBLISH]

{

if (!t\_newtran()) {

#absorb retransmissions

sl\_reply("500","Could not create transaction");

exit;

}

if (can\_publish\_reg("location")) {

$var(ret)= publish\_reg("location");

switch ($var(ret)){

case 1: #success

xlog("L\_DBG", "Publish reg successful");

break;

case -1: #failure

xlog("L\_ERR", "Publish reg failure - sending 500 Error now\n");

t\_reply("500","Server Error publishing subscription");

break;

default:

xlog("L\_ERR", "Unknown return code from publish reg event alue is [$var(ret)]\n");

break;

}

} else {

t\_reply("403","Forbidden to PUBLISH");

exit;

}

}

######################################################################

# Subscribe route

######################################################################

route[SUBSCRIBE]

{

if (!t\_newtran()) {

#absorb retransmissions

sl\_reply("500","Could not create transaction");

exit;

}

if (!has\_totag()) {

xlog("L\_DBG", "This is an initial SUBSCRIBE\n");

if (!term\_impu\_registered("location")) {

xlog("L\_DBG", "We need to do an UNREG server SAR assignment\n");

assign\_server\_unreg("SUBSCRIBE\_UNREG\_SAR\_REPLY", "location", "term");

exit;

}

if (!can\_subscribe\_to\_reg("location")){

t\_reply("403","Forbidden to SUBSCRIBE");

exit;

}

}else{

xlog("L\_DBG", "This is a subsequent SUBSCRIBE\n");

}

$var(ret)= subscribe\_to\_reg("location");

switch ($var(ret)){

case 1: #success

xlog("L\_DBG", "Subscribe to reg successful");

break;

case -1: #failure

xlog("L\_ERR", "Subscribe to reg failure - sending 500 Error now\n");

t\_reply("500","Server Error saving subscription");

break;

case -2: #error

xlog("L\_ERR", "Subscribe to reg error sending notify - 200 OK so subscription already sent\n");

break;

default:

xlog("L\_ERR", "Unknown return code from subscribe to reg event alue is [$var(ret)]\n");

break;

}

}

route[SUBSCRIBE\_UNREG\_SAR\_REPLY]

{

xlog("L\_DBG","saa\_return code is $avp(s:saa\_return\_code)\n");

switch ($avp(s:saa\_return\_code)){

case 1: #success

xlog("L\_DBG", "SAR success - will process subscribe\n");

if (can\_subscribe\_to\_reg("location")){

$var(ret)= subscribe\_to\_reg("location");

switch ($var(ret)){

case 1: #success

xlog("L\_DBG", "Subscribe to reg successful");

break;

case -1: #failure

xlog("L\_ERR", "Subscribe to reg failure - sending 500 Error now\n");

t\_reply("500","Server Error saving subscription");

break;

case -2: #error

xlog("L\_ERR", "Subscribe to reg error sending notify - 200 OK so subscription already sent\n");

break;

default:

xlog("L\_ERR", "Unknown return code from subscribe to reg event alue is [$var(ret)]\n");

break;

}

}else{

t\_reply("403","Forbidden to SUBSCRIBE");

exit;

}

break;

case -1: #failure

xlog("L\_ERR", "SAR failure - Sending 403 Forbidden\n");

t\_reply("403","Forbidden to SUBSCRIBE");

break;

case -2: #error

xlog("L\_ERR", "SAR error - Sending 403 Forbidden\n");

t\_reply("403","Forbidden to SUBSCRIBE");

break;

default:

xlog("L\_ERR", "Unknown return code from SAR, value is [$avp(s:saa\_return\_code)] - sending 403 Forbidden\n");

t\_reply("403","Forbidden to SUBSCRIBE");

break;

}

exit;

}

######################################################################

# XMLRPC routing

######################################################################

#!ifdef WITH\_XMLRPC

route[XMLRPC] {

if ((method=="POST" || method=="GET")

#!ifdef XMLRPC\_WHITELIST\_1

&& ((src\_ip == XMLRPC\_WHITELIST\_1)

#!ifdef XMLRPC\_WHITELIST\_2

|| (src\_ip == XMLRPC\_WHITELIST\_2)

#!endif

#!ifdef XMLRPC\_WHITELIST\_3

|| (src\_ip == XMLRPC\_WHITELIST\_3)

#!endif

)

#!endif

) {

# close connection only for xmlrpclib user agents (there is a bug in

# xmlrpclib: it waits for EOF before interpreting the response).

if ($hdr(User-Agent) =~ "xmlrpclib")

set\_reply\_close();

set\_reply\_no\_connect();

dispatch\_rpc();

exit;

}

send\_reply("403", "Forbidden");

exit;

}

#!endif

######################################################################

# Route for handling Registrations:

######################################################################

route[REGISTER] {

if (!ims\_www\_authenticate(NETWORKNAME)) {

if ($? == -2) {

t\_reply("403", "Authentication Failed");

exit;

} else if ($? == -3) {

t\_reply("400", "Bad Request");

exit;

} else {

#user has not been authenticated. Lets send a challenge via 401 Unauthorized

xlog("L\_DBG","About to challenge! auth\_ims\n");

ims\_www\_challenge("REG\_MAR\_REPLY", "$td");

exit;

}

} else {

xlog("L\_DBG", "Auth succeeded\n");

# We need to check if this user is registered or not

if (!impu\_registered("location")) {

xlog("L\_ERR", "Not REGISTERED\n");

save("PRE\_REG\_SAR\_REPLY","location");

exit;

} else {

isc\_match\_filter\_reg("1","location");

save("REG\_SAR\_REPLY","location");

exit;

}

}

}

route[REG\_MAR\_REPLY]

{

#this is async so to know status we have to check the reply avp

xlog("L\_DBG","maa\_return code is $avp(s:maa\_return\_code)\n");

switch ($avp(s:maa\_return\_code)){

case 1: #success

xlog("L\_DBG", "MAR success - 401/407 response sent from module\n");

break;

case -1: #failure

xlog("L\_ERR", "MAR failure - error response sent from module\n");

break;

case -2: #error

xlog("L\_ERR", "MAR error - sending error response now\n");

t\_reply("500", "MAR failed");

break;

default:

xlog("L\_ERR", "Unknown return code from MAR, value is [$avp(s:uaa\_return\_code)]\n");

t\_reply("500", "Unknown response code from MAR");

break;

}

exit;

}

route[PRE\_REG\_SAR\_REPLY]

{

xlog("L\_DBG","saa\_return code is $avp(s:saa\_return\_code)\n");

#this is async so to know status we have to check the reply avp

xlog("L\_DBG","saa\_return code (for scscf\_save on register) is $avp(s:saa\_return\_code)\n");

switch ($avp(s:saa\_return\_code)){

case 1: #success

xlog("L\_DBG", "SAR success - 200 response sent from module\n");

isc\_match\_filter\_reg("0","location");

exit;

case -1: #failure

xlog("L\_ERR", "SAR failure - error response sent from module\n");

break;

case -2: #error

xlog("L\_ERR", "SAR error - error response sent from module\n");

break;

default:

xlog("L\_ERR", "Unknown return code from SAR, value is [$avp(s:uaa\_return\_code)]\n");

break;

}

exit;

}

route[REG\_SAR\_REPLY]

{

xlog("L\_DBG","saa\_return code is $avp(s:saa\_return\_code)\n");

#this is async so to know status we have to check the reply avp

xlog("L\_DBG","saa\_return code (for scscf\_save on register) is $avp(s:saa\_return\_code)\n");

switch ($avp(s:saa\_return\_code)){

case 1: #success

xlog("L\_DBG", "SAR success - 200 response sent from module\n");

exit;

case -1: #failure

xlog("L\_ERR", "SAR failure - error response sent from module\n");

break;

case -2: #error

xlog("L\_ERR", "SAR error - error response sent from module\n");

break;

default:

xlog("L\_ERR", "Unknown return code from SAR, value is [$avp(s:uaa\_return\_code)]\n");

break;

}

exit;

}

######################################################################

# Apply privacy, if requested

######################################################################

route[apply\_privacy]

{

if (is\_present\_hf("Privacy") && ($hdr(Privacy)=="id")) {

remove\_hf("P-Asserted-Identity");

}

}

######################################################################

# Originating, Intial Requests

######################################################################

route[orig]

{

xlog("L\_DBG","Enter orig route\n");

set\_dlg\_profile("orig");

# we MAYBE need something like this to check if a user is barred

# if (S\_originating\_barred()){

# sl\_send\_reply("403","Forbidden - Originating Public Identity barred");

# exit;

# }

if (is\_method("INVITE|SUBSCRIBE")) {

$avp(RR\_CUSTOM\_USER\_AVP)="mo";

record\_route();

}

# Start new transaction:

t\_newtran();

# check if dialog saved as fwded to AS

if (isc\_match\_filter("orig", "location")) {

t\_on\_failure("isc\_orig\_failure");

#xlog("Orig - msg was fwded to AS\n");

exit;

}

if (!isc\_from\_as("orig")) {

remove\_hf("P-Asserted-Identity");

append\_hf("P-Asserted-Identity: <sip:$fU@$fd>\r\n");

}

#!ifdef WITH\_RO

# before we allow call - lets check credit

if (is\_method("INVITE")) {

xlog("L\_DBG","Sending initial CCR Request for call\n");

$var(cc\_ret) = Ro\_CCR("CHARGING\_CCR\_REPLY", "orig", 30, "0", "0");

if ($var(cc\_ret) < 0) {

xlog("L\_ERR","CCR Request failure\n");

sl\_send\_reply("402","Payment required");

exit;

}

xlog("L\_DBG","CCR Request success\n");

exit;

}

#!endif

route(Finalize\_Orig);

}

route[Finalize\_Orig]

{

# Check for PSTN destinations:

if (is\_method("INVITE")) {

route(PSTN\_handling);

}

t\_on\_reply("orig\_reply");

t\_relay();

}

route[CHARGING\_CCR\_REPLY]

{

xlog("L\_DBG","saa\_return code is $avp(s:cca\_return\_code)\n");

switch ($avp(s:cca\_return\_code)){

case 1: #success

xlog("L\_DBG", "CCR success - will route message\n");

route(Finalize\_Orig);

break;

case -1: #failure

xlog("L\_ERR", "CCR failure - error response sent from module\n");

send\_reply("402","Payment required");

break;

case -2: #error

xlog("L\_ERR", "CCR error - error response sent from module\n");

send\_reply("500", "Charging Error");

break;

default:

xlog("L\_ERR", "Unknown return code from CCR: [$avp(s:cca\_return\_code)] \n");

send\_reply("500", "Charging Error");

break;

}

exit;

}

######################################################################

# Replies to the Initial Requests

######################################################################

onreply\_route[orig\_reply]

{

xlog("L\_DBG","Orig reply\n");

route(apply\_privacy);

break;

}

######################################################################

# Originating, subsequent requests

######################################################################

route[orig\_subsequent]

{

xlog("L\_DBG","Orig\_Subsequent\n");

if (!is\_method("ACK")) {

t\_on\_reply("orig\_subsequent\_reply");

}

t\_relay();

}

######################################################################

# Replies for originating, subsequent requests

######################################################################

onreply\_route[orig\_subsequent\_reply]

{

xlog("L\_DBG","Orig\_Subsequent\_reply\n");

route(apply\_privacy);

break;

}

######################################################################

# Failure-Route for Requests to an AS

######################################################################

failure\_route[isc\_orig\_failure]

{

xlog("L\_DBG","ISC\_Orig\_failure\n");

if (t\_check\_status("(408)|(5..)")){

t\_on\_failure("isc\_orig\_failure");

if (isc\_match\_filter("orig","location")){

xlog("L\_DBG","ISC\_Orig\_failure - msg was fwded to AS\n");

exit;

}

if (isc\_from\_as("origfail")) {

remove\_hf("P-Asserted-Identity");

append\_hf("P-Asserted-Identity: <sip:$fU@$fd>\r\n");

}

t\_on\_reply("orig\_reply");

t\_relay();

}

}

######################################################################

# Terminating requests

######################################################################

route[term]

{

xlog("L\_DBG","Term\n");

set\_dlg\_profile("term");

#we need something like this to check if a user is barred

# if (S\_terminating\_barred()){

# sl\_send\_reply("404","Not Found - Terminating user barred");

# exit;

# }

if (is\_method("INVITE|SUBSCRIBE")) {

$avp(RR\_CUSTOM\_USER\_AVP)="mt";

$avp(i:20)="mt";

record\_route();

}

# check if dialog saved as fwded to AS

if (isc\_match\_filter("term","location")){

t\_on\_failure("isc\_term\_failure");

xlog("L\_DBG","Term - msg was fwded to AS\n");

exit;

}

if (lookup("location")) {

if (uri=~"sip:(.\*)@"+NETWORKNAME\_ESC+"(.\*)") {

if (!t\_newtran()) {

sl\_reply\_error();

exit;

}

t\_reply("404","Not Found - destination user not found on this S-CSCF");

exit;

}

} else {

# User not registered? Reply with 404.

if (!t\_newtran()) {

sl\_reply\_error();

exit;

}

t\_reply("404","Not Found - destination user not found on this S-CSCF");

exit;

}

route(apply\_privacy);

t\_relay();

}

######################################################################

# Failure Route for Terminating requests

######################################################################

failure\_route[isc\_term\_failure]

{

xlog("L\_DBG","ISC\_term\_failure\n");

if (t\_check\_status("(408)|(5..)")){

t\_on\_failure("isc\_term\_failure");

if (isc\_match\_filter("term","location")){

xlog("L\_DBG","Term - msg was fwded to AS\n");

exit;

}

if (lookup("location")) {

if (uri=~"sip:(.\*)"+NETWORKNAME\_ESC+"(.\*)"){

t\_reply("404","Not Found - destination user not found on this S-CSCF");

exit;

}

} else {

t\_reply("404","Not Found - destination user not found on this S-CSCF");

exit;

}

t\_relay();

}

}

######################################################################

# Terminating, subsequent requests

######################################################################

route[term\_subsequent]

{

xlog("L\_DBG","term\_subsequent\n");

route(apply\_privacy);

t\_relay();

}

######################################################################

# Check for PSTN destinations:

######################################################################

route[PSTN\_handling]

{

# First, we translate "tel:"-URI's to SIP-URI's:

# $ru: tel:+(34)-999-888-777

# $fu: sip:test@foo.com

# becomes $ru: sip:+34999888777@foo.com;user=phone

if (!tel2sip("$ru", "$fd", "$ru"))

xlog("L\_WARN","Failed to convert $ru to a sip:-URI - M=$rm R=$ru F=$fu T=$tu IP=$si:$sp ID=$ci\n\n");

if ($rU =~ "\+[0-9]+") {

# Now let's check, if the number can be found in ENUM:

if(!enum\_query()) {

# ENUM failed, send it to the PSTN-Gateway:

route(PSTN);

break;

}

}

}

######################################################################

# Send calls to the PSTN-Gateways:

######################################################################

route[PSTN]

{

# if (!ds\_select\_domain("1", "4")) {

# xlog("L\_WARN","No PSTN-Gateways available - M=$rm R=$ru F=$fu T=$tu IP=$si:$sp ID=$ci\n\n");

# send\_reply("503", "Service not available");

# exit;

# }

# Relay the request:

t\_on\_failure("PSTN\_failure");

t\_relay();

exit;

}

######################################################################

# manage failure routing cases, perform failover

######################################################################

failure\_route[PSTN\_failure] {

# Choose another gateway, in case we

# - get a local generated "408"

# - receive a 5xx or 6xx reply from the proxy.

if (t\_branch\_timeout() || t\_check\_status("[5-6]..")) {

# if (ds\_next\_domain()) {

# Do Failover in case problems:

# t\_on\_failure("PSTN\_failure");

# t\_relay();

# } else {

# Add a header, to indicate the phone should try again in 30 seconds.

append\_hf("Retry-After: 30\r\n");

send\_reply("503", "Service not available");

}

exit;

}

#}