APPENDIX A

Sample Scripts

Note The scripts that appear in this section are only examples. They are not necessarily intended to be fully functional. For the latest versions of all sample scripts, go to the Developer Support web site at http://www.cisco.com/go/developersupport

SIP Headers

Passing SIP Headers

The following script prompts for an account number, then makes a call to URL "sip:elmo@sip.tgw.com," where the account number is passed in the SIP header under the header named "AccountInfo." Other static headers, such as Subject, To, From, and Priority, are also passed from the script, either as part of the URL or separately.

```
# prompt_digit_xfer.tcl
# Script Version: 2.0.0
#------------------------------------------------------------------
#
# Copyright (c) 2004 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------

proc init { } {
    global param

    set param(interruptPrompt) true
    set param(abortKey) *
    set param(terminationKey) #

}

proc act_Setup { } {

    leg setupack leg_incoming
    leg proceeding leg_incoming
    leg connect leg_incoming

    media play leg_incoming http://townsend.cisco.com/vxml/audio/enterAccount.au
}
```

```
proc act_MediaDone { } {
    global param

    set pattern(account) .+
    set callinfo(alertTime) 30
    leg collectdigits leg_incoming param pattern

}
proc act_GotAccount { } {
    global dest
    global callinfo
    global headers
    global account

    set status [infotag get evt_status]

    if {  $status == "cd_005" } {
    set account [infotag get evt_dcdigits]

    # These are additional headers for the setup request message
    set headers(AccountInfo) "[set account]"
    # this Subject header overwrites the one overloaded in the destination URL
    set headers(Subject) "HelloSipTCL"
    set headers(To) "sip:oscar@abc.com"
    set headers(From) "sip:nobody"
    set callinfo(protoHeaders) headers

    # this destination URL has an overloaded header named "Subject"
    set dest "sip:elmo@sip.tgw.com?Subject=Hello&Priority=Urgent"

    leg setup $dest callinfo leg_incoming

    } else {
    puts "\nCall [infotag get con_all] got event $status collecting destination"
    call close
    }
}

proc act_CallSetupDone { } {

    set status [infotag get evt_status]
    puts "\n STATUS=$status"
    if { $status == "ls_000"} {

        timer start leg_timer 30 leg_incoming
        return
    }
    call close
}

proc act_Cleanup { } {



    set status [infotag get evt_status]
        puts "\n STATUS is $status"

#    puts "\nCAME BACK FROM TRANSFER AND CLOSING CALL"
    call close
}

init
requiredversion 2.0
```

```
#----------------------------------
#   State Machine
#----------------------------------
  set fsm(any_state,ev_disconnected)          "act_Cleanup        same_state"
  set fsm(CALL_INIT,ev_setup_indication)      "act_Setup          MEDIAPLAY"
  set fsm(MEDIAPLAY,ev_media_done)            "act_MediaDone      GETDEST"
  set fsm(GETDEST,ev_collectdigits_done)      "act_GotAccount     PLACECALL"
  set fsm(PLACECALL,ev_setup_done)            "act_CallSetupDone  CALLACTIVE"
  set fsm(CALLACTIVE,ev_leg_timer)            "act_Cleanup        same_state"
  set fsm(CALLACTIVE,ev_disconnected)         "act_Cleanup        CALLDISCONNECT"
  set fsm(CALLDISCONNECT,ev_disconnected)     "act_Cleanup        same_state"
  set fsm(CALLDISCONNECT,ev_media_done)       "act_Cleanup        same_state"
  set fsm(CALLDISCONNECT,ev_disconnect_done)  "act_Cleanup        same_state"
  set fsm(CALLDISCONNECT,ev_leg_timer)        "act_Cleanup        same_state"

  fsm define fsm CALL_INIT
```

Retrieving SIP Headers

The following script plays the prompt "The number is," along with the account number received from the originating gateway. Other headers received are displayed in debug messages.

```
# prompt_digit_xfer.tcl
# Script Version: 2.0.0
#------------------------------------------------------------------
#
# Copyright (c) 2004 by cisco Systems, Inc.
# All rights reserved.
#------------------------------------------------------------------

proc init { } {
    global param

    set param(interruptPrompt) true
    set param(abortKey) *
    set param(terminationKey) #

}

proc act_Setup { } {
    leg setupack leg_incoming
    leg proceeding leg_incoming
    leg connect leg_incoming
    infotag set med_language 1

    set ani [infotag get leg_ani]
    set dnis [infotag get leg_dnis]

    puts "\n get_headers.tcl: ani is $ani"
    puts "\n get_headers.tcl: dnis is $dnis"

    set  Subject [infotag get evt_proto_headers Subject]
    puts "\n get_headers.tcl: Subject =  $Subject"

    set  Priority [infotag get evt_proto_headers Priority]
    puts "\n get_headers.tcl: Priority =  $Priority"

    set  From [infotag get evt_proto_headers From]
    puts "\n get_headers.tcl: From =  $From"
```

```
    set  To [infotag get evt_proto_headers To]
    puts "\n get_headers.tcl: To =  $To"

    set  Via [infotag get evt_proto_headers Via]
    puts "\n get_headers.tcl: Via =  $Via"

    set  tsp [infotag get evt_proto_headers tsp]
    puts "\n get_headers.tcl: tsp =  $tsp"

    set  phone_context [infotag get evt_proto_headers phone-context]
    puts "\n get_headers.tcl: phone-context =  $phone_context"

    set  AccountInfo [infotag get leg_proto_headers AccountInfo]
    puts "\n get_headers.tcl - act_Setup: AccountInfo =  $AccountInfo"

    media play leg_incoming tftp://townsend.cisco.com/audio/num_is.au %n$AccountInfo
}

proc act_CallSetupDone { } {

    set status [infotag get evt_status]
    puts "\n STATUS=$status"
    if { $status == "ls_000"} {

        timer start leg_timer 30 leg_incoming
        return
    }
    call close
}

proc act_Cleanup { } {

    set status [infotag get evt_status]
        puts "\n STATUS is $status"

#    puts "\nCAME BACK FROM TRANSFER AND CLOSING CALL"
    call close
}

init
requiredversion 2.0

#-----------------------------------
#   State Machine
#-----------------------------------
  set fsm(any_state,ev_disconnected)          "act_Cleanup        same_state"
  set fsm(CALL_INIT,ev_setup_indication)      "act_Setup          MEDIAPLAY"
  set fsm(PLACECALL,ev_setup_done)            "act_CallSetupDone  CALLACTIVE"
  set fsm(CALLACTIVE,ev_leg_timer)            "act_Cleanup        same_state"
  set fsm(CALLACTIVE,ev_disconnected)         "act_Cleanup        CALLDISCONNECT"
  set fsm(CALLDISCONNECT,ev_disconnected)     "act_Cleanup        same_state"
  set fsm(CALLDISCONNECT,ev_media_done)       "act_Cleanup        same_state"
  set fsm(CALLDISCONNECT,ev_disconnect_done)  "act_Cleanup        same_state"
  set fsm(CALLDISCONNECT,ev_leg_timer)        "act_Cleanup        same_state"

  fsm define fsm CALL_INIT
```

Services

The following script demonstrates how to use the service register command and the start service configuration. The script registers as a service to provide data to other scripts and wakes up on a timer to get the latest data.

Service Register and Start

```
#------------------------------------------------------------------
# sample_service.tcl -
# This script will not accept incoming calls.
#------------------------------------------------------------------

set myname "sample_service.tcl"

#------------
# Someone is telling us to terminate. The service will unregister
# when we close.
#
proc act_Terminate { } {
    global myname

    puts "$myname received a terminate event, closing up."
    call close
}

#------------
# Got an initial event when the script starts running. Register as
# as service, get the initial data, and wait on a timer.
#
proc act_Session { } {
    global myname

    puts "$myname is starting up."
    puts "Starting timer for 30 seconds"
    timer start call_timer0 30

    init_data

    set r [service register data_service]
    puts "  Register of data_service returned $r"
}

#-----------
# Simulate getting some data. We could subscribe, or be collecting info
# from calls. Just keep a counter.
#
proc init_data { } {
    global data

    set data(run_time) 0
    set data(call_count) 0
}

proc get_data { } {
    global data

    set data(run_time) [expr $data(run_time)+30]
}
```

```
#------------
# The initial call setup has come in. Output some display to the console.
# If we are the second session, send a message to the first.
#
proc act_Setup { } {
    global myname

    puts "$myname got a setup. Refusing the call."
    leg disconnect leg_all
}

#------------
# We received a message from another session.
#
proc act_Rx_Msg { } {
    global myname
    global data

    set src [infotag get evt_msg_source]
    infotag get evt_msg msg_array
    incr data(call_count)

    puts "$myname got a message. Respond with the data."
    set r [sendmsg $src -p data]
    puts "Send of data to $src returned $r"
}

#------------
# The timer has gone off. Get the data, and reset the timer.
#
proc act_Timer { } {
    puts "Timer went off. Get data, and reset the timer.\n"

    get_data
    timer start call_timer0 30
}

#------------
# Ignore this event. Output the event name to the console.
#
proc act_Ignore { } {
    global myname

    set ev [infotag get evt_event]
    puts "$myname is ignoring event $ev"
}


#---------------------------------
#   State Machine
#---------------------------------
#
  set fsm(any_state,ev_session_indication) "act_Session     same_state"
  set fsm(any_state,ev_session_terminate)  "act_Terminate   same_state"
  set fsm(CALL_INIT,ev_setup_indication)   "act_Setup       same_state"
  set fsm(any_state,ev_msg_indication)     "act_Rx_Msg      same_state"
  set fsm(any_state,ev_call_timer0)        "act_Timer       same_state"
  set fsm(any_state,ev_any_event)          "act_Ignore      same_state"

  fsm define fsm start_state
```

Session Interaction

The following script demonstrates session interaction. The script runs a VoiceXML script to get the list of handles.

```
#
# Global Data Structures
#    handles - list of handles of other sessions (not our own)
#              If they crash, we will not know, so may be out of date.
#    my_handle - handle of this session
#    my_leg - leg id of the initial leg we will keep
#    recorded_names - array of ptrs to RAM based recordings, indexed by handles.
#                     includes our own.
#

#------------
# The init routine runs once when the script is loaded.
# Setup the global media_parm array for use every time we turn on
# digit collect.
#
proc init { } {
    # Code here runs when the script is loaded.

    global media_parm

    puts "
              Loading si_demo_main.tcl

         "
    #
    # Setup parameter array for collecting digits
    #
    set media_parm(interruptPrompt) true
    set media_parm(enableReporting) true
    set media_parm(maxDigits) 1
}

#------------
#
# When the call comes in, just check in with the si demo
# server by firing up the vxml doc that will submit our handle.
#
# I expect it to return a ptr to a recorded name of this caller,
# and a list of active participants in the demo.
#
proc act_Setup { } {
    global my_handle
    global handles
    global my_leg

    set handles ""
    set my_handle [infotag get mod_handle]
    set my_leg [infotag get leg_incoming]
    puts "
         si_demo_main.tcl got a setup for leg $my_leg
             my_handle=$my_handle
         "

    leg setupack leg_incoming
    leg connect leg_incoming

    set parray(handle) [infotag get mod_handle]
    leg vxmldialog leg_incoming -u http://px1-sun/dramstha/si_demo.vxml -p parray
```

```
}

#------------
# Playout the list of participants so far.
#
proc play_list { } {
    global recorded_names

    puts " Playout the list of callers."

    foreach handle [array names recorded_names] {
    lappend playlist $recorded_names($handle)
    }
    if { [llength $playlist] <= 0 } {
    media play leg_incoming http://px1-sun/dramstha/no_others.au
    } else {
    media play leg_incoming http://px1-sun/dramstha/participants_are.au $playlist
    }
}

#------------
#
# Handoff the call to another leg.
#   We could setup a menu to select which one, but here we simply
#   find the first that is not us.
#
proc act_do_handoff { } {
    global handles
    global my_handle

    if { [llength $handles] <= 0 } {
        puts "Weird handofff, now no handles"
    media play leg_incoming http://px1-sun/dramstha/no_others.au
    fsm setstate RUNNING
    return
    }

    foreach handle $handles {
    set ret [handoff callappl leg_incoming $handle -s "Here is a leg"]
    if { $ret != "unavailable" } break;
    }
    if { $ret == "unavailable" } {
    puts "Bailing out on the handoff"
    media play leg_incoming http://px1-sun/dramstha/no_others.au
    fsm setstate RUNNING
    }
    puts "
       $my_handle handed off leg to $handle  status=$ret"
}

#------------
# We got a digit while conferenced. Don't care who sent it, disconnect.
#
proc act_Control_Conf { } {
    puts "Destroying the connection"
    connection destroy con_all
}

#------------
# Main routine to handle a digit when running the demo.
#   Implement the 4 options, playout help for anything else.
#
proc act_Control { } {
    global media_parm
```

```
    global handles

    set digit [infotag get evt_digit]
    puts "Digit entered=$digit"
    leg collectdigits leg_incoming media_parm

    if { [infotag get evt_state_current] != "RUNNING" } {
    puts "
        Ignoring digit when not running yet"
    return
    }

    switch $digit {
    1 {
            play_list
    }

    2 {
        if { [llength $handles] <= 0 } {
        media play leg_incoming http://px1-sun/dramstha/no_others.au
        } else {
        media play leg_incoming http://px1-sun/dramstha/sending_msg.au
        set oparms(id) msg
        foreach handle $handles {
            set r [sendmsg $handle -p oparms]
        }
        }
    }

    3 {
        if { [llength $handles] <= 0 } {
        media play leg_incoming http://px1-sun/dramstha/no_others.au
        } else {
                puts "Playing handing_off.au"
        media play leg_incoming http://px1-sun/dramstha/handing_off.au
        fsm setstate PRE_HANDOFF
        }
    }

    4 {
        media play leg_incoming http://px1-sun/dramstha/si_feature_description.au
http://px1-sun/dramstha/si_demo_description.au
    }

    default {
        media play leg_incoming http://px1-sun/dramstha/si_help.au
    }
    }
}

#------------
# Got a return from the initial submit to the si demo server.
#
# Save the recorded name of this caller, and the list of handles of other
# callers.
# Checkin with the other callers if there are any.
# Now we are up and running, although we may not have everyones
# name until we get responses from the checkin.
#
proc act_submit_done { } {
    global my_name
    global handles
    global media_parm
    global recorded_names
```

```
    global my_handle

    puts "Tcl script si_demo_main got a return from VXML"
    set my_name http://px1-sun/dramstha/si_unknown_caller.au
    set handles ""

    set r_event [infotag get evt_vxmlevent]
    puts "
          Tcl got on return from initial submit:
             evt_vxmlevent=$r_event
          "

    if { [string match error $r_event] == 1 } {
          return
    }
    set rlist [infotag get evt_vxmlevent_params parray]
    set msg "     Returned data from si server includes:"
    foreach i $rlist {
        append msg "\n            $i=$parray($i)\n"
    if { $i == "handles" } {
        set handles $parray($i)
        regsub $my_handle $handles "" $handles
        set index [lsearch $handles $my_handle]
    }
    if { $i == "recorded_name" } {
        set my_name $parray($i)
    }
    }
    puts "
        Returned data from si server includes: $msg"

    set recorded_names([infotag get mod_handle]) $my_name
    checkin
    #
    # We are up and running:
    puts "****RUNNING handles=$handles"
    leg collectdigits leg_incoming media_parm
    media play leg_incoming http://px1-sun/dramstha/si_help.au
}

#------------
# Checkin with other active calls. Send each a checking message.
#
proc checkin { } {
    global handles
    global my_name

    set oparms(recorded_name) $my_name
    set oparms(id) checkin
    foreach handle $handles {
        if { $handle == [infotag get mod_handle] } continue
    set r [sendmsg $handle -p oparms]
    puts "
            Just sent a checkin msg to $handle.
            Status=$r"
    }
}

#------------
# All done, disconnect the calls we are handling.
#
proc act_Cleanup { } {
    set ev [infotag get evt_event]
    puts "Script for callids <[infotag get leg_all]> got event $ev"
```

```
    puts "Closing now."
    call close
}

#------------
#
# Send a submit to remove the incoming leg from the database. Tell the other
# sessions (calls) we are gone.
#
proc act_db_remove { } {
    global handles

    puts "
         Tcl Firing up VXML to submit to remove handle [infotag get mod_handle]"
    set parray(handle) [infotag get mod_handle]
    leg vxmldialog leg_incoming -u http://px1-sun/dramstha/si_demo_disconnect.vxml -p
parray

    set oparms(id) "uncheckin"
    foreach handle $handles {
    set r [sendmsg $handle -p oparms]
    puts "Uncheckin message Send to $handle returned $r"
    }
}

#------------
#
# Got a message from another instance. Handle it:
#   If they are checking in, save the info and respond.
#   If they are responding to a checkin, save the info.
#   If their caller is sending a msg, tell our caller
#
proc act_rx_msg { } {
    global handles
    global recorded_names
    global my_name

    set src [infotag get evt_msg_source]
    set parm_list [infotag get evt_msg parm_array]

    set msg ""
    foreach i $parm_list {
        append msg "\n         parm($i)=$parm_array($i)"
    }
    puts "
      Leg [infotag get leg_incoming] got a message with [llength $parm_list] args
           from $src
           $msg"

    if { [info exists parm_array(id)] } {
    set id $parm_array(id)
    } else {
    set id "unknown"
    }

    if { [info exists parm_array(recorded_name)] } {
    set recorded_name $parm_array(recorded_name)
    } else {
    set recorded_name "http://px1-sun/dramstha/si_unknown_caller.au"
    }

    if { $id == "checkin" } {
        lappend handles $src
    set recorded_names($src) $recorded_name
```

```
        puts "\n  $src checked in. Now handles=
            $handles"
        set oparms(id) "checkin_rsp"
    set oparms(recorded_name) $my_name
        set r [sendmsg $src -p oparms]
        puts "Message Send to $src returned $r"
    }

    if { $id == "checkin_rsp" } {
        puts "\nSomeone responded to our check in"
    if { [lsearch $handles $src] == -1 } {
        puts "
               Error, We don't have src of the checkin response
                  src=$src
              handles=$handles"
    } else {
        set recorded_names($src) $recorded_name
    }
    }

    if { $id == "uncheckin" } {
    set index [lsearch handles $src]
    if { $index == -1 } {
        puts "
               Error, We don't know who is unchecking in
                  src=$src
              handles=$handles"
    } else {
        lreplace $handles $index $index
    }
    puts "
             After uncheckin, handles=$handles"
    }

    if { $id == "msg" } {
        puts "
          Telling our incoming leg we got a message"
        media stop leg_incoming
        media play leg_incoming http://px1-sun/dramstha/received_msg_from.au
$recorded_names($src)
    }
}

#------------
#
# Refuse a handoff of another leg by simply returning it.
#
proc act_fail_handoff { } {
    puts "
        Got a handoff in state [infotag get evt_state_current]. Returning it."
    set legs [infotag get leg_all]
    set leg2 [lindex $legs 1]
    handoff return $leg2 refused
}

#------------
#
# We got a handoff of another leg. Conference him
# in.
#
proc act_handoff { } {
    global leg1
    global leg2
    global my_leg
```

```
    puts "Got a handoff. string=[infotag get evt_handoff_string]"


    set legs [infotag get leg_all]
    set leg1 [lindex $legs 0]
    set leg2 [lindex $legs 1]

    if { [infotag get evt_state_current] != "RUNNING" } {
    puts "
        Returning handoff leg in state [infotag get evt_state_current]"
    handoff return $leg2 success
    fsm setstate RUNNING
    return
    }

    media stop $my_leg
    puts "Connecting legs $leg1 and $leg2 together"
    connection create $leg1 $leg2
}

#------------
#
# The conference create completed. Simply turn on collect digits so
# both callers can disconnect.
#
proc act_create_done { } {
    global leg1
    global leg2
    global media_parm

    puts "Connection Created."
    leg collectdigits $leg1 media_parm
    leg collectdigits $leg2 media_parm
}

#------------
#
# The conference has been destroyed. Return the other leg to it's
# original session.
#
proc act_destroy_done { } {
    global my_leg
    global leg1
    global leg2

    # Figure out which one to keep.
    if { $leg1 == $my_leg } {
    set retleg $leg2
    } else {
    set retleg $leg1
    }

    puts "
      connection destroyed.
          Returning leg $retleg, keeping $my_leg"
    handoff return $retleg "success"
    media play $my_leg http://px1-sun/dramstha/si_help.au
}

#------------
# We got the call leg returned to us after handoff.
# Just go back to running.
#
```

```
proc act_return { } {
    global media_parm

    puts "
        Leg was returned. Going back to running"
    media play leg_incoming http://px1-sun/dramstha/si_help.au
    leg collectdigits leg_incoming media_parm
}

#------------
# Ignore an event.
#
proc act_Ignore { } {
    set ev [infotag get evt_event]
    puts "
     Script for callids <[infotag get leg_all]>
       in state [infotag get evt_state_current]
       is ignoring event $ev"
}

init

#----------------------------------
#   State Machine
#----------------------------------
#
# States:
#    SUBMIT - We got a setup message, and submitted our handle to the
#             server.
#    RUNNING - Waiting for a command from the caller. Could be playing a
#              prompt.
#    HANDED_OFF - We handed off the incoming leg to another session. We
#                 have no legs here.
#    PRE_HANDOFF - playing a prompt to caller telling them we will handoff
#    CONFERENCED - Either setting up (waiting for conf create done) or
#                  actually conferenced.
#    UNCONFING   - Disconnecting the conference prior to returning a leg
#    DBREMOVE - Got a disconnect, removing ourselves from the server Data Base before
#               closing up shop.
#    CALLDISCONNECT - All done. Have disconnected the caller.
#    CALL_INIT - Initial state before a call comes in
#

  set fsm(any_state,ev_any_event)      "act_Ignore          same_state"
  set fsm(any_state,ev_disconnected)   "act_db_remove       DBREMOVE"
  set fsm(any_state,ev_msg_indication) "act_rx_msg          same_state"
  set fsm(any_state,ev_handoff)        "act_fail_handoff    same_state"
  set fsm(any_state,ev_create_done)    "act_create_done     same_state"
  set fsm(any_state,ev_destroy_done)   "act_destroy_done    RUNNING"
  set fsm(any_state,ev_digit_end)      "act_Ignore          same_state"
  set fsm(any_state,ev_media_done)     "act_Ignore          same_state"

  set fsm(RUNNING,ev_handoff)          "act_handoff         CONFERENCED"
  set fsm(CONFERENCED,ev_digit_end)    "act_Control_Conf    UNCONFING"
  set fsm(RUNNING,ev_digit_end)        "act_Control         same_state"
  set fsm(SUBMIT,ev_vxmldialog_done)    "act_submit_done    RUNNING"
  set fsm(DBREMOVE,ev_vxmldialog_done)  "act_Cleanup        CALLDISCONNECT"

  set fsm(CALL_INIT,ev_setup_indication) "act_Setup         SUBMIT"

  set fsm(CALLDISCONNECT,ev_disconnect_done) "act_Ignore    same_state"
  set fsm(PRE_HANDOFF,ev_media_done)     "act_do_handoff    HANDED_OFF"
  set fsm(HANDED_OFF,ev_returned)        "act_return        RUNNING"
  fsm define fsm CALL_INIT
```

Hybrid Scripting

The following script adds the message waiting indication(MWI) feature to an existing Tcl-VoiceXML (VXML) hybrid application that places an outbound call. The MWI feature is implemented using the SIP SUBSCRIBE/NOTIFY mechanism.

The script demonstrates how to separate the VXML portion to handle the IVR and the Tcl portion to handle call control. The VXML dialogs will play prompts and collect inputs as well as interfaces with the HTTP web server. The main Tcl script handles the call control portion which provides the equivalent function of the standard VXML Transfer. When the user calls into the gateway, a VXML dialog is invoked. The VXML dialog queries the backend server with the ANI of the caller. The VXML document returned presents the user with choices of who he can call. This can be the personal phone book or a company directory. In addition, the VXML Transfer element attributes are passed from the VXML document to the Tcl procedure to place call. The returned information also includes a URL which contains the location of VXML dialog to run after the called party hangs up. This VXML dialog emulates a voicemail message which is played to the user if he chooses to listen to them.

The user also hear a prompt indicating new voicemails in his mailbox after the call has been connected. He can choose to listen to the voicemail or ignore it. If he selects to listen to his voicemail, the music on hold treatment will be applied to the called party This feature is enabled when the Tcl application subscribes to a MWI event package running on a SIP server. For this sample, the MWI event package is another Tcl application running on another gateway. The server application will notify the client application when a new message is delivered to the mailbox.

```
#-------------------------------------------------------------------------------------
# Procedure init
#
# The init procedure defines the initial parameters of digit collection. Users are
# expected to enter a single DTMF key within 5 secs and are allowed to enter a digit
# before a prompt is played out. This is used when a prompt is played to the user
# indicating a new voicemail and the user may enter any DTMF key to listen to the
# voicemail message.

proc init { } {

    global param

    set param(maxDigits) 1
    set param(interruptPrompt) true
    set param(initialDigitTimeout) 5
}

#-------------------------------------------------------------------------------------------
# Procedure init_Config
#
# In this procedure, CLI configurations are read
# Users can configure the location of the Tcl script on the router and this is stored in
# the variable, baseURI.
# Users can configure the number of notifications that are sent before the subscription
# closes. The default is 4.
#

proc init_ConfigVars {} {

    global notNum
    global baseURI
```

```
   if {[infotag get cfg_avpair_exists base-url]} {
        set baseURI [string trim [infotag get cfg_avpair base-url]]
   } else {
        set baseURI tftp://TFTP-SERVER/scripts/
   }

   if {[infotag get cfg_avpair_exists notification-num]} {
        set notNum [string trim [infotag get cfg_avpair notification-num]]
   } else {
        set notNum 4
   }

    puts "\n\n\n**** Base Url is $baseURI\n\n"
    puts "\n\n\n**** Number of notifications in CLI is $notNum\n\n"
}


#-------------------------------------------------------------------------------------------
# Procedure init_perCallVars
#
# In this procedure, the global variables are initialized
# The IP address for the SIP server is configured on the router as follows :
#  ip host x.x.x.x sip-server

proc init_perCallVars { } {

    global ani
    global uri
    global menu
    global name
    global mNew
    global subId
    global mTotal
    global bridge
    global subUrl
    global subInfo
    global counter
    global tclProc
    global maxtime
    global mWaiting
    global callActive
    global longpound
    global destination
    global message_status
    global message_summary
    global connectiontimeout

    set uri ""
    set name ""
    set bridge ""
    set baseURI ""
    set tclProc ""
    set maxtime ""
    set longpound ""
    set destination ""
    set connectiontimeout ""

    set counter 0
    set callActive 0
    set subUrl sip:mwi@sip-server

}

#-------------------------------------------------------------------------------------------
```

```
# Procedure act_Start
#
# The act_Start procedure is executed when it receives an ev_setup indication event.
# A setup acknowledgement, call proceeding and connect message is sent to the incoming
# call leg. The parameters in the subscribe request are defined and a subscribe request
# to the MWI event package is sent.
#

proc act_Start { } {

    global ani
    global subId
    global subUrl
    global subInfo
    global baseURI

    init_perCallVars
    infotag set med_language 1

    leg setupack leg_incoming
    leg proceeding leg_incoming
    leg connect leg_incoming

    set ani [infotag get leg_ani]

    set subInfo(event) mwi
    set subInfo(notificationReceiver) mwiClient
    # expiration time is 1 hour
    set subInfo(expirationTime) 3600
    set subinfo(subscriptionContext) context
    set headers(Account) $ani
    set headers(Subject) "Message Waiting Indication Subscription"
    set subInfo(protoHeaders) headers
    set subInfo(protoContents) "This is a subscription request from mwiclient"
    set subInfo(protoContentTypes) "text/plain"

    set subId [subscription open $subUrl subInfo]

}

#-------------------------------------------------------------------------------------
# Procedure act_VxmlDialog1
#
# This procedure is called from act_Notify. It is called once in the application after the
# script receives a ev_notify_ind event.
# In this procedure, the ANI of the user is submitted to sipWebQuery.php script on the
# backend web server. The result is a dynamically generated VXML document which contains
# information required for call setup. The user is prompted to enter the destination
# number of the person he wishes to call. The destination number together with the
# attributes to the VXML transfer element and the URL for the subsequent VXML dialog are
# passed back to the Tcl script. The IP address of the Web server is configured on the
# router as follows: ip host x.x.x.x HTTP-SERVER

proc act_VxmlDialog1 { } {

    global baseURI

    puts "\n\n\n******** Procedure  VxmlDialog1  ********\n\n"

    set vxmlDialog1 {
        <vxml version="2.0">
            <form id="main">
                <catch event="error.badfetch.com">
                    <log>Web Server down ! Submit action in VxmlDialog1 failed. </log>
```

```
                    <exit/>
                </catch>

                <var name="WEB_SERVER" expr="'http://HTTP-SERVER/vxml/'"/>
                <var name="ANI" expr="session.telephone.ani"/>

                <block>
                <submit expr="WEB_SERVER+'sipWebQuery.php'" method="get" namelist="ANI"/>
                </block>

            </form>
        </vxml> }

    leg vxmldialog leg_incoming -u $baseURI -v $vxmlDialog1
    fsm setstate WEBQUERY
}

#-------------------------------------------------------------------------------------------
# Procedure act_SubscribeDone
#
# This procedure is executed when an ev_unsubscribe_done event is received. The status of
# the event is checked. If the subscription failed, the subscription is closed.

proc act_SubscribeDone { } {


    global subId

    puts "\n\n\n******** Procedure  SubscribeDone  ********\n\n"

    set subId [infotag get evt_subscription_id]

    puts "\n\n\n**** Subscription ID is $subId"
    set status [infotag get evt_status]

    switch $status {
        "sn_000" {
        puts "\n\n\n**** Subscription status, $status swith ID: $subId is successful\n\n"
                    }
        "sn_001" {
        puts "\n\n\n**** Subscription status, $status with ID: $subId is pending\n\n"
                    }
        "sn_002" {
        puts "\n\n\n**** Subscription statu, $status swith ID: $subId failed\n\n"
        subscription close $subId
                    }
    }
}

#-------------------------------------------------------------------------------------------
# Procedure act_Notify
#
# This procedure is received when an ev_notify_ind event is received. Both standard and
# non standard SIP headers sent by the server are read. The non standard headers are
# MessageStatus and MessageSummary. MessageStatus indicates that there are new
# voicemail and MessageSummary indicates the number of new voicemails.
# An acknowledgement is sent back to the server in response to the notification event.
# This procedure is called every time an ev_notify_ind event is received (except for the
# last one). The subscription is terminated depending of the value configured
# for the number of times a notification is sent.

proc act_Notify { } {

    global ani
```

```
global mNew
global mTotal
global notNum
global counter
global mWaiting

puts "\n\n\n******** Procedure Notify  ********\n\n"


set status [infotag get evt_status]
puts "\n\n\n**** Status of subscription is $status\n\n"

set subId [infotag get evt_subscription_id]
puts "\n\n\n**** Subscription ID is subId=$subId"

set event_header [infotag get evt_proto_headers Event]
puts "\n\n\n****  Event header is $event_header\n\n"

set hello [infotag get evt_proto_headers "Hello"]
puts "\n\n\n**** Hello Header from Notification is $hello\n\n"

set message_status [infotag get evt_proto_headers "MessageStatus"]
regexp {Messages-waiting:([a-z]+)} $message_status dummy mWaiting
puts "\n\n\n**** Message Waiting is $mWaiting\n\n"

set message_summary [infotag get evt_proto_headers "MessageSummary"]
regexp {Voicemail:([0-9]+)} $message_summary dummy mNew
puts "\n\n\n**** New message is $mNew\n\n"

set content_type [infotag get evt_proto_content_type]
puts "\n\n\n**** Notification content_type=$content_type\n\n"

set content [infotag get evt_proto_content]
puts "\n\n\n**** Notification content=$content\n\n"

set exp_time [infotag get evt_proto_headers Expires]
puts "\n\n\n**** Subscription Expires: $exp_time\n\n"

set notifyRecr [infotag get subscription_info $subId notificationReceiver]
puts "\n\n\n**** Notification Receiver is  $notifyRecr\n\n"


# subscription expired

if {$exp_time == 0} {

    puts "\n\n\n**** Subscription is terminated"

    set headers(Title) "This is the last acknowledgement from the client"
    set ackInfo(protoContents) "Ending subscription"
} else {
    set headers(Title) "This is an acknowledgement from the client"
    set ackInfo(protoContents) "This is CONTENT from client"
}

set headers(Account) $ani
set ackInfo(protoHeaders) headers
set ackInfo(respCode) ack
set ackInfo(protoContentTypes) "text/plain"

subscription notify_ack $subId -i ackInfo

if {$counter == 0} {
    puts "\n\n\n**** Start VXML Dialog \n\n"
```

```
            act_VxmlDialog1
            incr counter
            return
        }

        if {$counter < $notNum} {
              if {$mWaiting == "yes"} {
                   act_MessageIndication
                   incr counter
               }
              return
        }

        puts "\n\n\n**** Number of notifications received is $counter"

        # close the subscription
        puts "\n After $counter notifications, close the subscription now.\n\n"

        subscription close $subId
        fsm setstate CLOSESUB
    }

    #-------------------------------------------------------------------------------------
    # Procedure act_MessageIndication
    #
    # This procedure is first invoked when the script receives an ev_leg_timer event. The
    # timer event comes up if there is a new voicemail after the call has been connected.
    # To play a prompt to the user that a new voicemail is in the mailbox, the connection
    # between the incoming and outgoing call legs is destroyed.
    #

    proc act_MessageIndication {} {

        global mNew
        global param
        global mTotal
        global mWaiting
        global callActive

        puts "\n\n\n******** Procedure MessageIndication ********\n\n"

        if {$callActive == 1} {
            set callActive 0
            connection destroy con_all
        } else {
            leg collectdigits leg_incoming param

            media play leg_incoming _new_voicemail.au %n$mNew %s1000 _check_new_voicemail.au
        }
        fsm setstate CHECKVOICEMAIL
    }

    #-------------------------------------------------------------------------------------
    # Procedure act_PlayMessageInd
    #
    # This procedure plays the prompt "You have X new voicemails" to the user when the script
    # receives an ev_destroy_done event is received. The act_PlayMusic is invoked to provide
    # music on hold treatment to the called party

    proc act_PlayMessageInd {} {

        global mNew
        global param
        global mTotal
```

```
    global mWaiting

    puts "\n\n******* Procedure PlayMessageInd ********\n\n"
    puts "\n\n\n**** Number of new vmail are $mNew\n\n"

    leg collectdigits leg_incoming param
    media play leg_incoming _new_voicemail.au  %n$mNew %s1000 _check_new_voicemail.au
        #media play leg_incoming _you_have.au %n$mNew %s1000 _seconds.au

    act_PlayMusic

    fsm setstate CHECKVOICEMAIL
}

#-------------------------------------------------------------------------------------------
# Procedure act_PlayMusic
#
# This procedure plays a music prompt to the called party while the user is checking
# his voicemail messages.
#
proc act_PlayMusic {} {

    puts "\n\n******* Procedure PlayMusic ********\n\n"

    media play leg_outgoing _song.au
    fsm setstate MUSICONHOLD
}

#-------------------------------------------------------------------------------------------
# Procedure act_FinalNotify
#
# This procedure is invoked when the script receives the final ev_notify_ind event.
# The SIP server application sends this event when the client script closes the
# subscription.

proc act_FinalNotify { } {

    puts "\n\n******* Procedure FinalNotify ********\n\n"

    set status [infotag get evt_status]
    puts "\n\nstatus=$status\n\n"
    set sub_id [infotag get evt_subscription_id]

    # you can access any standard headers
    set From [infotag get evt_proto_headers From]
    puts "\n\n**** From header is: $From\n\n"

    set headers(Title) "Hello, this is the final acknowledgement from the client"
    set ackInfo(protoHeaders) headers
    set ackInfo(protoContents) "This is CONTENT from client"
    set ackInfo(protoContentTypes) "text/plain"
    set ackInfo(respCode) ack

    subscription notify_ack $sub_id -i ackInfo
}

#-------------------------------------------------------------------------------------------
# Procedure act_UnsubscribeDone
#
# This procedure is invoked when the script receives an ev_unsubscribe_done event. This
# event is sent when the subscription is closed.

proc act_UnsubscribeDone { } {
```

```
    puts "\n\n\n******** Procedure UnsubscribeDone  ********\n\n"

    set status [infotag get evt_status]
    puts "\n\n\n**** Status of unsubscribe_done is $status\n\n"

}

#-------------------------------------------------------------------------------------
# Procedure act_SubscribeClose
#
# This procedure is invoked when the script receives an ev_unsubscribe_indication or an
# ev_subscribe_cleanup event. An ev_unsubscribe_indcation event is received if the server
# sends unsubscribe request, subscription  expires or other errors. An
# ev_subscribe_cleanup event is received when the CLI, "clear sub all" is executed
# The subscription is closed.
#

proc act_SubscribeClose { } {

    puts "\n\n\n******** Procedure SubscribeClose  ********\n\n"

    puts "\n\n**** Unsubscribe Indication received or Subscribe Clean up event
    received\n\n"
    set sub_id [infotag get evt_subscription_id]
    subscription close $sub_id
}


#-------------------------------------------------------------------------------------
# Procedure act_GetTransferAttr
#
# This procedure handles the transfer attributes sent by VxmlDialog1.
# It parses the evParam array to get the attributes.

proc act_GetTransferAttr {} {

    global uri
    global name
    global bridge
    global evParam
    global tclProc
    global maxtime
    global longpound
    global destination
    global transfer_param
    global connectiontimeout

    puts "\n\n\n******** Procedure GetTransferAttr  ********\n\n"

    set tclProc GetTransferAttr
    set ev [infotag get evt_vxmlevent]

    if {$ev != "vxml.dialog.transferEvent"} {
        puts "\n\n\t\t ****  Expected event vxml.dialog.transferEvent, got $ev"
    }

    # get all the parameters sent from VXML dialog

    set eventParam [infotag get evt_vxmlevent_params evParam]

    for {set i 0} {$i < [llength $eventParam]} {incr i} {
        set [set value [lindex $eventParam $i]] $evParam($value)
        puts "\n\n\t\t **** $value = $evParam($value)"
    }
```

```
}

#-------------------------------------------------------------------------------------
# Procedure act_Transfer
#
# This procedure is called when the application receives the vxml_dialog_done event.
# It checks for the destination number, set the callInfo array with the relevant
# transfer attributes and places a call to that number.

proc act_Transfer { } {

    global uri
    global name
    global bridge
    global tclProc
    global maxtime
    global longpound
    global destination
    global transfer_param
    global connectiontimeout

    puts "\n\n\n******** Procedure Transfer ********\n\n"

    set tclProc Transfer

    # check the sub-event name
    set exp_ev vxml.session.complete
    set ev [infotag get evt_vxmlevent]

    if {$ev != $exp_ev} {
        puts "\n\n\t\t **** Expected event $exp_ev, got $ev"
    }

     # check the dialog status
     set status [infotag get evt_status]

     switch $status {
                    "vd_000" {
                              #get the transfer attribute, destination
                              infotag get evt_vxmlevent_params transfer_param

                              if !{[info exists transfer_param(destination)]} {
                                  puts "\n\n\t\t ****  destination number does not exist"
                                  act_LogStatus $status $tclProc
                              }

                              set destination $transfer_param(destination)
                              set callinfo(alerTime) $maxtime
                              set callinfo(newguid) $longpound
                              leg setup $destination callInfo leg_incoming
                    }

                  "vd_001" -
                  "vd_002" -
                  "vd_003" {
                           puts "\n\n\t\t ****  VXML Dialog status, expected vd_000, got
                          $status"
                           act_LogStatus $status $tclProc
                  }
                }
}

#-------------------------------------------------------------------------------------
# Procedure act_TransferDone
```

```
#
# If leg setup is successful, the 2 parties are conferenced. If not, the status of leg
# setup is sent to the web server. If there are new voicemail messages, a timer is
# started for 2 seconds and act_MessageIndication procedure is called.

proc act_TransferDone { } {

    global mNew
    global mTotal
    global tclProc
    global mWaiting
    global incoming
    global outgoing
    global callActive
    global destination
    global transferStatus

    puts "\n\n\n******** Procedure TransferDone  ********\n\n"

    set tclProc TransferDone
    set status [infotag get evt_status]

    puts "\n\n\t\t **** Status of leg setup is $status \n"
    switch $status {
        "ls_000" {
            set incoming [infotag get leg_incoming]
            set outgoing [infotag get leg_outgoing]
            set  creditTimeLeft [infotag get leg_settlement_time leg_all]

            if { ($creditTimeLeft == "unlimited") || ($creditTimeLeft=="uninitialized")}
                {puts "\n\n\t\t **** UnLimited Call Time\n"
                    set callActive 1
                    puts "\n\n\t\t **** Are there new voicemail messages ? $mWaiting \n\n"
                    if { $mWaiting == "yes" } {timer start leg_timer 2 leg_incoming}}
                    else {
                                # start the timer for ...
                                if { $creditTimeLeft < 10 } {
                                   set beep 1
                                   set delay $creditTimeLeft
                                } else {
                                set delay [expr $creditTimeLeft - 10]
                                }
                                timer start leg_timer $delay leg_incoming
                           }
                  }

            "ls_007" {puts "\n\n\t\t **** Call status is $status, Destination is Busy \n"
                set transferStatus NOANSWER
                act_LogStatus $transferStatus $tclProc
                  }
            "ls_008" {puts "\n\n\t\t **** Call status is $status, Incoming Disconnected
                \n"
                set transferStatus NEAR_END_DISCONNECT
                act_LogStatus $transferStatus $tclProc
                  }
                 "ls_009" {puts "\n\n\t\t **** Call status is $status, Outcoming Disconnect
                \n"
                set transferStatus FAR_END_DISCONNECT
                act_LogStatus $transferStatus $tclProc
                 }
                 default {
                           puts "\n\n\t\t **** Call status is $status\n"
                           set transferStatus UNKNOWN
                           act_LogStatus $transferStatus $tclProc
```

```
                }
    }
}

#-------------------------------------------------------------------------------------
# Procedure act_VxmlDialog2
#
# This procedure is called when the caller chooses to listen to his new voicemail
# messages, the script receives an ev_collectdigits_done event.
# After the called party hangs up, the application invokes the second VXML dialog which
# is a fake voicemail service for the user.

proc act_VxmlDialog2 { } {

    global uri
    global menu
    global baseURI
    global incoming
    global outgoing
    global callActive

    puts "\n\n\n******** Procedure VxmlDialog2  ********\n\n"
    puts "\n\n\t\t **** URL of voicemail is $uri \n\n"

    if {$callActive == 1} {
        leg disconnect leg_outgoing
        set callActive 0
    }

    set event [infotag get evt_event]
    set status [infotag get evt_status]

    switch $status {
                    "cd_005" {
                              puts "\n\n\n**** Caller wants to check new voicemail \n\n"
                              if {$uri != ""} {
                                  leg vxmldialog leg_incoming -u $uri
                              } else {
                                  # play media to say cannot access voicemails
                                  media play leg_incoming _technicalProblem.au
                              }

                    }
                    "cd_001" {
                              puts "\n\n\n**** Caller didn't enter digit to check new
voicemail\n\n"
                              media stop $outgoing

                              if {$callActive == 0} {
                                  connection create $incoming $outgoing
                                  set callActive 1
                                  return
                               }
                    }
                    "cd_010" {
                            puts "\n\n\n**** The digit collection was terminated because
                           of an unsupported or unknown feature or event\n\n"
                             return
                    }
    }
}

#-------------------------------------------------------------------------------------
# Procedure act_VxmlDialog2Done
```

```
#
# This procedure is invoked when the ev_vxmldialog_done event is received.
# The conference between the incoming and outgoing call legs is bridged again after the
user is done listening to the new voicemail messages.

proc act_VxmlDialog2Done { } {

    global tclProc
    global incoming
    global outgoing
    global callActive

    puts "\n\n\n******** Procedure VxmlDialog2Done  ********\n\n"

    set tclProc VxmlDialog2Done
    set exp_ev vxml.session.complete
    set ev [infotag get evt_vxmlevent]

    if {$ev != $exp_ev} {
        puts "\n\n\t\t **** Expected event $exp_ev, got $ev"
        act_LogStatus $ev $tclProc
    }

    if {[infotag get evt_legs] == $incoming} {
        media stop $outgoing
        if {$callActive == 0} {
           connection create $incoming $outgoing
           set callActive 1
           return
        }

   }
}

#-------------------------------------------------------------------------------------------
# Procedure act_LogStatus
#
# The status code for leg setup, vxml dialog are sent to the backend web server in this
# procedure.

proc act_LogStatus {statusCode tclProcedure} {

    global baseURI

    puts "\n\n\n******** Procedure LogStatus  ********\n\n"
    puts "\n\n\t\t***** Status Code is $statusCode in procedure $tclProcedure"

    set vxmlDialog3 {
        <vxml version="2.0">
            <form id="main">

                <var name="WEB_SERVER" expr="'http://HTTP-SERVER/vxml/'"/>
                <var name="ANI" expr="session.telephone.ani"/>
                <var name="STATUSCODE" expr="com.cisco.params.code"/>
                <var name="PROCEDURE" expr="com.cisco.params.procedure"/>

                <catch event="error.badfetch.com">
                    <log>Web Server down ! Submit action in VxmlDialog3 failed. </log>
                    <exit/>
                </catch>

                <block>
                    <log> Tcl Status Code : <value expr="STATUSCODE"/> found in Tcl
                    Procedure : <value expr="PROCEDURE"/></log>
```

```
                    <submit expr="WEB_SERVER+'status.php'" method="get" namelist="ANI
                    STATUSCODE PROCEDURE"/>
                </block>

            </form>
        </vxml> }

    set tclStatusParam(code) $statusCode
    set tclStatusParam(procedure) $tclProcedure

    leg vxmldialog leg_incoming -u $baseURI -v $vxmlDialog3 -p tclStatusParam
    fsm setstate LOGSTATUS

}

#-------------------------------------------------------------------------------------------
# Procedure act_Timer
#
# This procedure is invoked when the timer expires. If the call duration is unlimited,
# this will not be invoked.

proc act_Timer { } {

    global beep
    global incoming
    global outgoing

    puts "\n\n\n******** Procedure Timer  ********\n\n"

    set incoming [infotag get leg_incoming]
    set outgoing [infotag get leg_outgoing]

    if { $beep == 0 } {
        #insert a beep ...to the caller
        connection destroy con_all
        set beep 1
    } else {
        connection destroy con_all
        fsm setstate LASTWARN
    }
}

proc act_LastWarn { } {
    media play leg_incoming _outOfTime.au
}

proc act_Destroy { } {
    media play leg_incoming _beep.au
}

#-------------------------------------------------------------------------------------------
# Procedure act_PlayMessageIndDone
# When the script receives an ev_media_done event is received, this procedure will be
# executed. If the event happened on the outgoing leg, the music prompt is repeated to
# the called party. If the event is received on the incoming leg,  the prompt playout
# is terminated on the outgoing side and the 2 legs are  conferenced.

proc act_PlayMessageIndDone { } {

    global callActive
    global incoming
    global outgoing

    puts "\n\n\n******** Procedure PlayMessageIndDone  ********\n\n"
```

```
    set incoming [infotag get leg_incoming]
    set outgoing [infotag get leg_outgoing]

    if {[infotag get evt_legs] == $outgoing} {
         media play leg_outgoing _song.au
         return
    }
    if {[infotag get evt_legs] == $incoming} {
        media stop $outgoing
        if {$callActive == 0} {
           connection create $incoming $outgoing
           set callActive 1
           return
        }
    }
}
proc act_Beeped { } {

    global incoming
    global outgoing

    connection create $incoming $outgoing
}

proc act_ConnectedAgain { } {

    puts "\n\n\t\t***** Call Connected Again\n\n "
}

#-------------------------------------------------------------------------------------
# Procedure act_HandleOutgoing
#
# When the called party hangs up, the connection is destroyed

proc act_HandleOutgoing { } {

    global outgoingDisconnect

    puts "\n\n\t\t********* Procedure HandleOutgoing *********\n\n "

    if {[infotag get evt_legs] == [infotag get leg_outgoing]} {

        # Outgoing disconnected

        connection destroy con_all
        set outgoingDisconnect 1
    } else {
        call close
        fsm setstate CALLDISCONNECT
    }
}

#-------------------------------------------------------------------------------------
# Procedure act_LongPound
#
# If the user enters the Pound key, the connection to the called party will be destroyed.
# The user hear the voicemail emulation in VxmlDialo2

proc act_LongPound { } {

    puts "\n\n\n******** Procedure LongPound  ********\n\n"
    puts "\n\n\t\t***** Calling Party  entered long pound"
```

```
    set DURATION 1000

    if {[infotag get evt_digit] != "#"} {
        fsm setstate same_state
    } else {
        set duration [infotag get evt_digit_duration]

        if {$duration < $DURATION} {
            fsm setstate same_state
        } else {
            connection destroy con_all
        }
    }
}

#-------------------------------------------------------------------------------------------
# Procedure act_Cleanup
#
# This procedure is invoked when the application receives an ev_disconnected event.

proc act_Cleanup { } {

    puts "\n\n\n******** Procedure Cleanup  ********\n\n"
    call close
}

#-------------------------------------------------------------------------------------------
# Procedure act_SessionClose
#
# This procedure will be invoked if a script receives an ev_session_terminate event.
# Note that this will not be invoked in this application since this application requires
an incoming call leg

proc act_SessionClose { } {

    puts "\n\n\n******** Procedure SessionClose  ********\n\n"
    call close
}

proc act_Ignore { } {
# Dummy Procedure
    puts "Event Capture"
}


requiredversion 2.0
init

init_ConfigVars

#---------------------------------
#   State Machine
#---------------------------------
  set fsm(any_state,ev_disconnected)             "act_Cleanup               same_state"
  set fsm(any_state,ev_subscribe_done)             "act_SubscribeDone         SUBSCRIBED"
  set fsm(any_state,ev_notify_ind)                 "act_Notify                NOTIFIED"
  set fsm(any_state,ev_unsubscribe_done)           "act_UnsubscribeDone       same_state"
  set fsm(any_state,ev_unsubscribe_indication)     "act_SubscribeClose        same_state"
  set fsm(any_state,ev_subscribe_cleanup)          "act_SubscribeClose        same_state"
  set fsm(any_state,ev_session_terminate)          "act_SessionClose          same_state"

  set fsm(CALL_INIT,ev_setup_indication)           "act_Start                 SUBSCRIBE"
  set fsm(NOTIFIED,ev_media_done)                  "act_Ignore                same_state"
  set fsm(CHECKVOICEMAIL,ev_collectdigits_done)    "act_VxmlDialog2           same_state"
```

```
set fsm(MUSICONHOLD,ev_media_done)              "act_PlayMessageIndDone    same_state"
set fsm(CLOSESUB,ev_notify_ind)                 "act_FinalNotify           same_state"

set fsm(WEBQUERY,ev_vxmldialog_event)           "act_GetTransferAttr       same_state"
set fsm(WEBQUERY,ev_vxmldialog_done)            "act_Transfer              TRANSFER"
set fsm(TRANSFER,ev_setup_done)                 "act_TransferDone          CALLACTIVE"
set fsm(CALLACTIVE,ev_leg_timer)              "act_MessageIndication     CHECKVOICEMAIL"
set fsm(CALLACTIVE,ev_disconnected)           "act_HandleOutgoing        CONNDESTROY"
set fsm(CALLACTIVE,ev_digit_end)              "act_LongPound             CONNDESTROY"
set fsm(CHECKVOICEMAIL,ev_vxmldialog_done)    "act_VxmlDialog2Done       same_state"
set fsm(CHECKVOICEMAIL,ev_create_done)        "act_ConnectedAgain        CALLACTIVE"
set fsm(LOGSTATUS,ev_vxmldialog_done)         "act_Cleanup               same_state"
set fsm(CHECKVOICEMAIL,ev_destroy_done)       "act_PlayMessageInd        same_state"
set fsm(PLAYBEEP,ev_media_done)               "act_Beeped                same_state"
set fsm(INSERTBEEP,ev_destroy_done)           "act_Destroy               same_state"
set fsm(INSERTBEEP,ev_media_done)             "act_Beeped                same_state"
set fsm(INSERTBEEP,ev_create_done)            "act_ConnectedAgain        CALLACTIVE"
set fsm(LASTWARN,ev_destroy_done)             "act_LastWarn              CALLDISCONNECT"
set fsm(CALLDISCONNECT,ev_disconnect_done)    "act_Cleanup               same_state"
set fsm(CALLDISCONNECT,ev_media_done)         "act_Cleanup               same_state"
set fsm(CALLDISCONNECT,ev_disconnect_done)    "act_Cleanup               same_state"
set fsm(CALLDISCONNECT,ev_leg_timer)          "act_Cleanup               same_state"

fsm define fsm CALL_INIT
```