Tcl IVR Version 2.0 uses Tcl scripts to gather data and to process accounting information. For example, a Tcl IVR script can play an audio prompt that asks callers to enter a specific type of information, such as a personal identification number (PIN). After playing the audio prompt, the Tcl IVR application collects the predetermined number of touch tones and sends the collected information to an external server for caller authentication and service authorization.

Guide Contents
Troubleshooting Cisco IOS Voice Overview
Debug Command Output on Cisco IOS Voice Gateways
Filtering Troubleshooting Output
Cisco VoIP Internal Error Codes
Troubleshooting Cisco IOS Voice Telephony
Troubleshooting Cisco IOS Voice Protocols
Troubleshooting Cisco IOS Telephony Applications
Monitoring the Cisco IOS Voice Network
Cause Codes and Debug Values

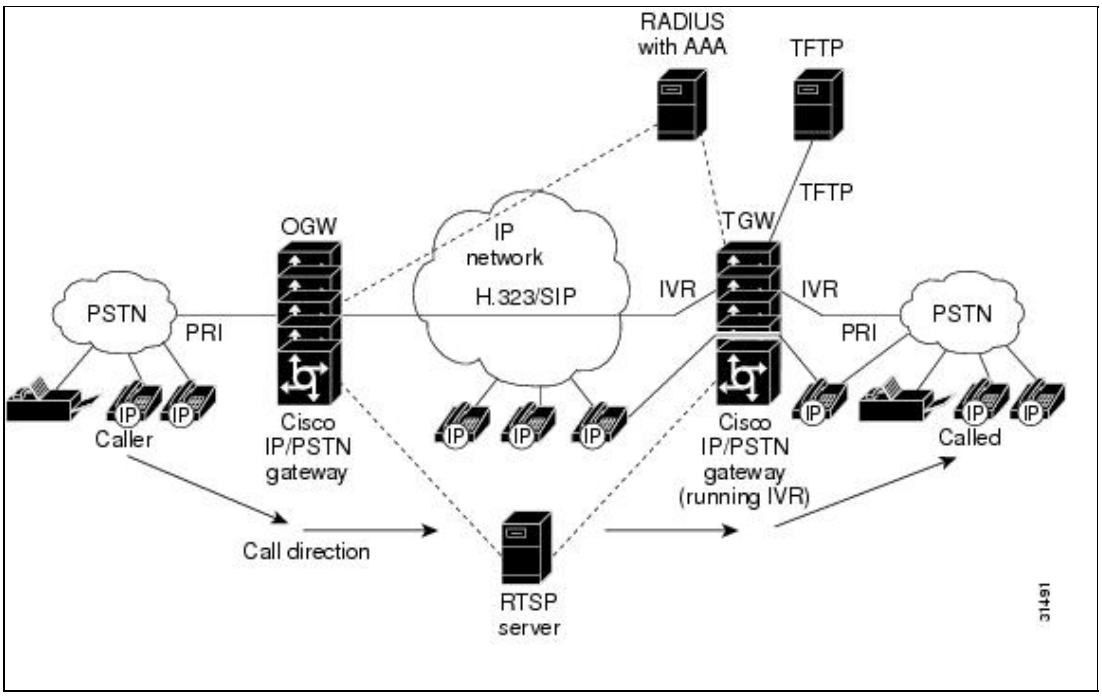
Contents

- 1 IVR Call Leg
- 2 Testing and Debugging Your Script
- 3 Loading Your Script
- 4 Associating Your Script with an Inbound Dial Peer
- 5 Displaying Information About IVR Scripts
- 6 Using URLs in IVR Scripts
    - 6.1 URLs for Loading the IVR Script
    - 6.2 URLs for Loading Audio Files
- 7 Tips for Using Your Tcl IVR Script

IVR Call Leg

Figure: IVR Control of Tcl Scripts on an IP Call Leg displays a Tcl IVR application on the gateway.

Figure: IVR Control of Tcl Scripts on an IP Call Leg



For information on developing Tcl scripts for voice applications, refer to the TCL IVR API Version 2.0 Programmer's Guide.

Testing and Debugging Your Script

It is important to thoroughly test a script before it is deployed. To test a script, you must place it on a router and place a call to activate the script. When you test your script, make sure that you test every procedure in the script and all variations within each procedure.

You can view debugging information applicable to the Tcl IVR scripts that are running on the router. The **debug voip ivr** command allows you to specify the type of debug output you want to view. To view debug output, enter the following command in privileged-EXEC mode:

```
[no] debug voip ivr {states | error | tclcommands | callsetup | digitcollect | script |
dynamic | applib | settlement | all}
```

For more information about the **debug voip ivr** command, see the Cisco IOS Debug Command Reference.

The output of any Tcl **puts** commands is displayed if script debugging is on.

Possible sources of errors are:

- An unknown or misspelled command (for example, if you misspell media play as mediaplay)
- A syntax error (such as, specifying an invalid number of arguments)
- Executing a command in an invalid state (for example, executing the **media pause** command when no prompt is playing)
- Using an information tag (info-tag) in an invalid scope (for example, specifying evt_dcdigits when not handling the ev_collectdigits_done event).

In most cases, an error such as these causes the underlying infrastructure to disconnect the call legs and clean

up.

Loading Your Script

To associate an application with your Tcl IVR script, use the following command:

```
(config)# call application voice application_name script_url
```

After you associate an application with your Tcl IVR script, use the following command to configure parameters:

```
(config)# call application voice application_name script_url [parameter value]
```

In this command:

- application_name specifies the name of the Tcl application that the system is to use for the calls configured on the inbound dial peer. Enter the name to be associated with the Tcl IVR script.
- script_url is the pathname where the script is stored. Enter the pathname of the storage location first and then the script filename. Tcl IVR scripts can be stored in Flash memory or on a server that is acceptable using a URL, such as a TFTP server.
- parameter value allows you to configure values for specific parameters, such as language or PIN length.

For more information about the **call application voice** command, refer to the Interactive Voice Response Version 2.0 on Cisco VoIP Gateways document.

In the following example, the application named "test" is associated with the Tcl IVR script called newapp.tcl, which is located at tftp://keyer/debit_audio/:

```
(config)# call application voice test tftp://keyer/debit_audio/newapp.tcl
```

Note: If the script cannot be loaded, it is placed in a retry queue and the system periodically retries to load it. If you modify your script, you can reload it using only the script name: (config)# call application voice load script_name

For more information about using the **call application voice** and **call application voice load** commands, refer to the Cisco IOS Tcl IVR and VoiceXML Application Guide. For more information about these commands, refer to the Cisco IOS Voice Command Reference.

Associating Your Script with an Inbound Dial Peer

To invoke your Tcl IVR script to handle a call, you must associate the application configured with an inbound dial peer. To associate your script with an inbound dial peer, enter the following commands in configuration mode:

```
(config)# dial-peer voice number voip
(conf-dial-peer)# incoming called-number destination_number
(conf-dial-peer)# application application_name
```

In these commands:

- number uniquely identifies the dial peer. (This number has local significance only.)

- destination_number specifies the destination telephone number. Valid entries are any series of digits that specify the E.164 telephone number.
- application_name is the abbreviated name that you assigned when you loaded the application.

For example, the following commands indicate that the application called "newapp" should be invoked for calls that come in from an IP network and are destined for the telephone number of 125.

```
(config)# dial-peer voice 3 voip
(conf-dial-peer)# incoming called-number 125
(conf-dial-peer)# application newapp
```

For more information about inbound dial peers, refer to Dial Peer Configuration on Voice Gateway Routers and the Cisco IOS Tcl IVR and VoiceXML Application Guide.

Displaying Information About IVR Scripts

To view a list of the voice applications that are configured on the router, use the **show call application voice** command. A one-line summary of each application is displayed.

```
show call application voice {name | summary}
```

In this command:

- *name* indicates the name of the desired IVR application. If you enter the name of a specific application, the system supplies information about that application.
- **summary** indicates that you want to view summary information. If you specify the summary keyword, a one-line summary is displayed about each application. If you omit this keyword, a detailed description of the specified application is displayed.

The following is example output of the **show call application voice** command:

```
Router# show call application voice session2
Idle call list has 0 calls on it.
Application session2
    The script is read from URL tftp://dirt/sarvi/scripts/tcl/app_session.tcl
    The uid-len is 10               (Default)
    The pin-len is 4                (Default)
    The warning-time is 60          (Default)
    The retry-count is 3            (Default)
    It has 0 calls active.
The TCL Script is:
------------------
# app_session.tcl
#-----------------------------------------------------------------
# Copyright (c) 1998, 1999 by cisco Systems, Inc.
# All rights reserved.
#-----------------------------------------------------------------
#
# This tcl script mimics the default SESSION app
#
#
# If DID is configured, just place the call to the dnis
# Otherwise, output dial-tone and collect digits from the
# caller against the dial-plan.
#
# Then place the call. If successful, connect it up, otherwise
# the caller should hear a busy or congested signal.
# The main routine just establishes the state machine and then exits.
```

```
# From then on the system drives the state machine depending on the
# events it receives and calls the appropriate tcl procedure
#---------------------------------
#   Example Script
#---------------------------------
proc init { } {
    global param
    set param(interruptPrompt) true
    set param(abortKey) *
    set param(terminationKey) #
}
proc act_Setup { } {
    global dest
    global beep
    set beep 0
    leg setupack leg_incoming
    if { [infotag get leg_isdid] } {
        set dest [infotag get leg_dnis]
        leg proceeding leg_incoming
        leg setup $dest callInfo leg_incoming
        fsm setstate PLACECALL
    } else {
        playtone leg_incoming tn_dial
        set param(dialPlan) true
        leg collectdigits leg_incoming param
      }
}
proc act_GotDest { } {
    global dest
    set status [infotag get evt_status]
    if {  $status == "cd_004" } {
        set dest [infotag get evt_dcdigits]
        leg proceeding leg_incoming
        leg setup $dest callInfo leg_incoming
    } else {
        puts "\nCall [infotag get con_all] got event $status while placing an outgoing
call"
        call close
    }
}
proc act_CallSetupDone { } {
    global beep
    set status [infotag get evt_status]
    if { $status == "CS_000"} {
        set  creditTimeLeft [infotag get leg_settlement_time leg_outgoing]
        if { ($creditTimeLeft == "unlimited") ||
             ($creditTimeLeft == "uninitialized") } {
            puts "\n Unlimited Time"
        } 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
        }
    } else {
        puts "Call [infotag get con_all] got event $status collecting destination"
        call close
    }
}
proc act_Timer { } {
```

```
    global beep
    global incoming
    global outgoing
    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 {
        media play leg_incoming flash:out_of_time.au
        fsm setstate CALLDISCONNECTED
    }
}
proc act_Destroy { } {
    media play leg_incoming flash:beep.au
}
proc act_Beeped { } {
    global incoming
    global outgoing
    connection create $incoming $outgoing
}
proc act_ConnectedAgain { } {
    timer start leg_timer 10 leg_incoming
}
proc act_Ignore { } {
# Dummy
    puts "Event Capture"
}
proc act_Cleanup { } {
    call close
}
init
#---------------------------------
#   State Machine
#---------------------------------
  set TopFSM(any_state,ev_disconnected) "act_Cleanup,same_state"
  set TopFSM(CALL_INIT,ev_setup_indication) "act_Setup,GETDEST"
  set TopFSM(GETDEST,ev_digitcollect_done) "act_GotDest,PLACECALL"
  set TopFSM(PLACECALL,ev_setup_done)  "act_CallSetupDone,CALLACTIVE"
  set TopFSM(CALLACTIVE,ev_leg_timer) "act_Timer,INSERTBEEP"
  set TopFSM(INSERTBEEP,ev_destroy_done) "act_Destroy,same_state"
  set TopFSM(INSERTBEEP,ev_media_done)  "act_Beeped,same_state"
  set TopFSM(INSERTBEEP,ev_create_done)    "act_ConnectedAgain,CALLACTIVE"
  set TopFSM(CALLACTIVE,ev_disconnected)   "act_Cleanup,CALLDISCONNECTED"
  set TopFSM(CALLDISCONNECTED,ev_disconnected) "act_Cleanup,same_state"
  set TopFSM(CALLDISCONNECTED,ev_media_done)  "act_Cleanup,same_state"
  set TopFSM(CALLDISCONNECTED,ev_media_done)  "act_Cleanup,same_state"
  set TopFSM(CALLDISCONNECTED,ev_disconnect_done) "act_Cleanup,same_state"
  set TopFSM(CALLDISCONNECTED,ev_leg_timer) "act_Cleanup,same_state"
  fsm define TopFSM  CALL_INIT
```

Using URLs in IVR Scripts

With IVR scripts, you use URLs to call the script and to call the audio files that the script plays. The VoIP system uses Cisco IOS File System (IFS) to read the files, so any IFS-supported URLs can be used, which includes TFTP, FTP, or a pointer to a device on the router.

Note: There is a limit of 32 entries in Flash memory, so you may not be able to copy all your audio files into Flash memory.

URLs for Loading the IVR Script

The URL of the IVR script is a standard URL that points to the location of the script. Examples include:

- flash:myscript.tcl-The script called myscript.tcl is being loaded from Flash memory on the router.
- slot0:myscript.tcl-The script called myscript.tcl is being loaded from a device in slot 0 on the router.
- tftp://BigServer/myscripts/betterMouseTrap.tcl-The script called myscript.tcl is being loaded from a server called BigServer in a directory within the tftpboot directory called myscripts.

URLs for Loading Audio Files

URLs for audio files are different from those used to load IVR scripts. With URLs for audio files:

- For static prompts, you can use the IFS-supported URLs as described in the URLs for Loading the IVR Script.
- For dynamic prompts, the URL is created by the software, using information from the parameters specified for the **media play** command and the language CLI configuration command.

Tips for Using Your Tcl IVR Script

This section provides some answers to frequently asked questions about using Tcl IVR scripts.

- How do I get information from my RADIUS server to the Tcl IVR script?

  After you have performed an authentication and authorization, you can use the **infotag get** command to obtain the credit amount, credit time, and cause codes maintained by the RADIUS server.

- What happens if my script encounters an error?

  When an error is encountered in the script, the call is cleared with a cause of TEMPORARY_FAILURE (41). If the IVR application has already accepted the incoming call, the caller hears silence. If the script has not accepted the incoming call, the caller might hear a fast busy signal.
  If the script exits with an error and IVR debugging is on (as described in the Testing and Debugging Your Script), the location of the error in the script is displayed at the command line.