NAME

PhoneScript – A language for building telephone based interactive voice response applications

SYNOPSIS

PhoneScript <file> [args...]

DESCRIPTION

PhoneScript runs the file *file* as a PhoneScript program. The PhoneScript language is a superset of TCL, the Tool Command Language system by John Ousterhout, University of California at Berkeley. PhoneScript programs contain TCL commands augmented by the PhoneScript specific commands described below.

The only syntactic addition to TCL is the notion of a keyword value. Any time an argument to one of the PhoneScript specific commands is of the form: *name=value*, with no spaces anywhere in the string, the parameter *name* is set to *value* for the duration of the command. Without the keyword parameter, the value of the TCL variable of the same name is used instead. This feature is permits the program to set global default values for many important parameters, yet override them on a case by case basis.

USAGE

When a PhoneStation program starts, *file* is read in and processed. Calls to the *on* command cause the body of the *on* expression to be assigned to the array "on", with an array index matching the specified on condition. The expressions are executed when the appropriate condition occurs. The TCL variable "on(current)" contains the name of the on condition currently executing.

At startup, all of the Unix environment variables are copied to the TCL variables of the same name, and the TCL variables *argc* and *argv* are set to the arguments supplied on the PhoneScript command line. The variable *pid* is set to the current process id.

The audio device (/dev/audio) is opened, and configured properly for use with the telephone interface. The telephone interface (see *stim())* is opened, run through a self test, and configured ready to place an outgoing call. If only incoming calls are required, the main body of the program should return, and the *on call* program will be run for each incoming call.

COMMANDS

In addition to the TCL commands described in the TCL reference manual, the following commands are added to provide access to the telephone. Any keywords used by the commands are indicated.

adsi clear

adsi define [id message]...

adsi assign id...

The *adsi* command is a preliminary interface for PhoneScript that permits PhoneStation to function as the SPCS (host) side of an ADSI telephone connection as described in the Bellcore technical advisory TA-NWT-001273. The commands return "1" if successful, and "0" otherwise. Invalid parameters cause *adsi* to abort with an appropriate error message. The *clear* option resets the ADSI telephone to its default state. The *define* option permits text strings (up to 16 characters each) to be assigned any one of the 32 "definers" (numbered 2-33), as described in the technical advisory. As many definers as can fit in a 254 character message may be sent at once. The *assign* option takes up to 8 parameters, each of which is a "definer" previously defined with an "adsi define" command. The first parameter corresponds to soft key 1, the second to key 2, etc, for each of the 8 keys. The definer "1" unmaps a soft key, causing it to return no value to the host. Soft keys on the telephone return three touch tones "Dnn", where "D" is the DTMF tone of the same name, and "nn" is the two digit number of the text currently associated with the soft key. The tones are returned only if there is text associated with the key. The *adsi* command is expected to change.

Audio The audio command provides low level access to the audio device. It is not normally required.

Audio drain

Wait for any pending audio to drain

Audio play <n>

Audio record <n>

Set the play or record volume. Normally this is not needed, as the default values are chosen to be reasonable. The parameter *n* varies from 1 to 255, with the default of 50. Values in excess of 100 not only cause distortion, but may destroy the telephone interface box.

beep Play a beep to the telephone. Someday there will be additional arguments that control the pitch and duration of the beep.

call <number> [<progress>] [dial_tone=n] [sharp=t] [wait_rings=rings]

Place a call to the phone number *number*, and wait up to *wait_rings* rings for the phone to be answered. Number is a string containing the digits 0-9, # to wait for an additional dialtone, and '-' for a 1/2 second pause. If the TCL program *progress*, if specified, is called to monitor the progress of the call. The variables *reason* and *ring* Contain a message indication the current progress state, and the number of audible rings heard. *progress is called when either ring* or *reason* changes. The keyword *dial_tone* specifies the time to wait for a dial tone before giving up. The default is 80, which is 8 seconds. If *sharp* is given, the single touch tone "t" is played when the called party answers the call.

cnv2tt <string>

Convert *string* to the numeric touch tone equivalents. Any ASCII character that is not on the telephone keypad is ignored.

db <option> [args]

The **db** command implements a simple relational file management system. It operates on single relation at a time stored as an ASCII file. The first row of the file contains semicolon terminated attribute names. The remaining rows contain semicolon terminated values. The current implementation limits the usable size of a database to several thousand lines. An example database is:

```
name;room;phone;
stephen;2B393;x4308;
mike;2A385;x4070;
mike;2A385;x4071;
ron;2D393;x3317;
```

The TCL variables whose names match the attributes of the database are used for transferring data between TCL and the database. The various database option either set the TCL variable from the database, or modify the values of the database to reflect the current values of the TCL variable. A description of the *db* options follows.

db close

Close the current database, and free up all resources associated with it. Close DOES NOT imply *db write*.

db delete

Delete the current row of the current database. The next row is made the current row.

db fields

Return the attribute names for the current database. This will also be the list of TCL variable names that are used to exchange information between the TCL program and the database.

db insert

Create a new row after the currently selected row. Its values are derived from the TCL variables that match the attribute names for the database.

db next Make the next row in the database the current row, setting the appropriate TCL variables. If there is no next row, *db next* returns "0", and sets all the variable values to the empty

string.

db process <expr>
For each selected tuple in the current database, the TCL program *expr*. is run. The values of the TCL variables that match the attribute names of the database are changed at each invocation to reflect the attribute values of the current row. The TCL commands *break* and *continue* may be used in the normal way within *expr*.

db select [and|or|set] <expr>
Mark as selected those rows in the database for which the TCL expression *expr* is true. If *and* is specified, only currently selected rows are examined. The rows that are not currently selected, remain NOT selected. If *or* is specified, only currently NOT selected rows are examined. Currently selected rows remain selected. If *set* (the default) is specified, all rows are evaluated by *expr*. The *expr* is run for each row in the database that is examined. In addition to the TCL variables that match the database item names, *NR* is set to the current rown number, and *selected* is true if the row is currently selected. The number of selected rows is returned, and alse left in the variable *select_count.*

db set <db_name>
Set the current database to the file named *db_name*. "1" is returned if a new database was opened, "2" if the database was previously opened, and is just being made the current database, and "0" if the database could not be opened. All lines in the database must have the same number of delimiters (;). Invalid lines are ignored, and cause *db set* to return an error. Although several database files may be opened at one time, only on can be "current".

db top Set the top of the database. An *insert* command will create a new first row, and a *next* command will make the first row the of the database the current row;

db update
Change the current row of the database to reflect the values in the equivalent TCL variables. If the new value for the entire row is the same size as the old value, then *update* is more efficient.

db write [<db_name>]
Write the current database back to a file. if *db_name* is specified, it is used as the database name instead of the name used to open the database with *db set* .

debug Enter debug mode. The user is prompted for PhoneScript commands at the terminal. Debug mode is terminated by typing exit. If the variable *debug* is set, then debug mode is automatically entered if a TCL command generates an error.

echo args...
Echo the arguments on stdout, separated by a space. If the keyword *nonl=yes* appears as an argument, no newline is written.

hangup Hangup the phone (if it is off hook), terminate the current TCL program, run the *on hangup* expression (if available), and wait for the next call to arrive.

on <condition> [<expression>]
Sets the named on condition to *expression*, the TCL program to be executes when the condition arises. If no expresion is specified, the action associated with the condition is cleared.

on call <expr>
When an incoming call is answered, the TCL program *expr* is run. The variable *calls* contains the number of calles answered so far.

on endringing <expr>
If the calling party hangs up before the minimum number of rings has transpired, the *expr* program is run. The variable *last_ring* is the number of rings counted.

on hangup <expr>

When the calling party terminates the phone call, or if the *hangup* command is issued, run the TCL program *expr*.

on int <expr>

If the PhoneScript program is interrupted, the TCL program *expr* is run before PhoneScript exits. **On int** can be used to clean up before exiting. The variable *temp_names* contains the list of temporary files created by *synth* that will be deleted.

on ringing <expr>

When the phone start to ring, the TCL program <*expr*> is run. The TCL variable *rings* may be modified at this time to indicate how many rings to wait before the phone is answered, and the *on call* program (if any) is run. If the variable *caller_id* is set to 1, and the Caller ID interface is available, then *number* is set to the telephone number of the calling party, as a 10 digit string.

on signal <expr>

If the program receives a SIG_USR1 signal while it is waiting for a call to arrive, <*expr*> is run. This mechanism permits other programs to reconfigure the program while it is running.

phone <option> [args]

The *phone* command permits low level access to the telephone interface. Each *option* controls one of the telephone interface functions. The options are described separately below.

phone answer

Wait for the called party to answer an outgoing call. This is used primarily to place outgoing calls. *Answer* listens for audible ringing, voice sounds, and busy signals, then guesses when a person answers the phone.

phone blink_fast

phone blink_slow

These two commands cause the phone interface box signal light to blink fast or slow respectively when the interface is on-hook. The default is *blink_slow*. These commands can be used to provide the user a visual indication of the state of a particular PhoneScript program.

phone close

Close the telephone interface. This is used primarily for debugging. The interface is set to its default values, and all other phone commands are disabled until a *phone open* command is issued.

phone dcd_off

Disable the DCD pin on the phone RS232 interface. Further reads form the phone interface will fail.

phone dcd_on

Enable the DCD pin on the phone RS232 interface.

phone dial <number>

Dial the phone number *number*. It is up to the program to insure the phone is off-hook at this point.

phone dtmf_off

Disable hardware touch tone detection.

phone dtmf_on

Enable hardware touch tone detection. This is the default.

phone flash

Simulate a "flash-hook" operation, by placing the phone-on hook for 0.5 seconds, then back off-hook. This only works if the phone interface is already in the on-hook state.

phone flush

Flush any phone data (i.e. touch tones) that may have been sent by the phone interface, but not yet received by the program.

phone loop_off
Turn off loop current detection. This is normally done while placing outgoing calls.

phone loop_on
Turn on loop current detection. This is the default. Loop current detection is normally enabled once a call is established, as the primary means of determining when the call is completed.

phone off_hook
Place the phone in the off-hook state.

phone on_hook
Place the phone in the on-hook state.

phone open [answer]
The phone interface is opened. If *answer* is specified, open waits until a phone call arrives before returning.

phone reset
Reset the entire telephone interface. Normally this should never be required. It might be necessary to close, then re-open the phone interface at this point.

phone tones <tones> [<volume>]
Play the tone s *tones* out the telephone interface, where *n* is one of "123456789∗0#ABCD". If touch tone detection is enabled at this point, the tone will be detected. *Volume* may be used to specify the tone volume, in percent of maximum. The default is 80.

phone status
Return the current status of the telephone interface. Status consists of the string: "adhrelp", where each letter is either upper or lower case. The upper case letter indicates ON, while the lower case letter indicates OFF. The letters represent: a– the phone interface is on, d– DCD is enabled, h– the phone is on hook, r– The phone is ringing, e– touch tones are enables, l– loop current

phone version
Prints the current version of the telephone interface software.

play [<files>] until <regex> unless <expr> [<options>]

play [<files>] until <regex} [<options>]

play [<files>] <regex> <expr> [<options>]

play [<files>] <regex> [<options>]

play [<file>] [<options>]
All of the variations of *play*, play zero or more audio files to the telephone, and gather touch tones from the user. All variants of *play* have some or all of the clauses: *files, until, unless*, or *options.*

Files is a list of audio files to play to the user. The file named "#" is special. It causes the next audio message in the **synth** queue to be played. The files are played sequentially until a touch tone is received from the user, which terminates the message output.

until is a regular expression that is compared against the touch tones accumulated so far by *play*. When the tones received match the regular expression, *regex*, **play** is done, and returns the tones. Normally the first tone received causes the *files* to stop playing, but **play** continues to listen for touch tones until either the time limit has expired (see *timelimit* below), or the received tones match the regular expression *regex.*

unless The *unless* expression, if supplied, is a TCL program that gets run each time a new touch

tone is received. A number of elements of the TCL array, *unless* are set as the *unless* program begins, which indicate the current state of **play.** These variables may be modified by the *unless* program, to effect changes in the way play operates. The array elements that are currently exchanged between *play* and the *unless* program are:

digits The number of tones accumulated so far (not including the one just typed).

done All of the files are done playing, and the *timeout* timer has been set. This variable is read-only, and may not be modified in the *unless* program.

maxtones
The maximum number of tones that may be entered before *play* terminates.

timelimit
the maximum time (in 10th of seconds) that play will continue accepting tones after all files have finished playing (either because a tone was entered, or the files ran out.)

hi_water
The maximum number of audio samples permitted in the audio output Q. I'll explain what this does later.

reason Reason is a character string containing one or more of the characters "tfdx", that indicates the conditions under which the *unless* expression is called. The letters "tfdx" stand for touch tones, a file finished playing, all files are finished playing, and the time limit expired, respectively.

file The index of the currently playing file in the list of files to play.

file_name
The name of the currently playing file, or blank, if no files are playing.

tone The just-received touch tone.

tones The list of touch tones received so far, not including the tone just received.

ignore Tones that are to be completely ignored when keyed in by the user.

file_pos the *seek* position of the currently playing file.

files The list of files to play.

unless_count
counts the number of times an unless expression was entered.

why Contains the four letters "tfdx", and represents the reason the unless program was run. The four letter stand for **t**ones, **f**iles, **D**one, and timelimit-e**X**pired respectively. If the letter is capitalized, then the associated action has occurred, causing the unless expression to be run. Normally *why* would be "Tfdx", indicating that a touch-tone has been received.

If either *file* or *files* are modified, then *play* stops playing the current file, and re-initializes the entire file list and current file. In some cases this cause play to resume after it has been stopped. In such cases the *timeout* parameter is also reset.

The elements of the *unless* array are not populated until they are referenced, so the TCL command "array names unless" won't return the list of possible array elements. Instead, it will return the single element *variables* which is a list containing all of the possible elements of *unless.* There are many other subtle effects of the *play* command that have yet to be defined, especially for cases where a play command with an *unless* clause is nested in another *unless clause.* When the *unless* program is finished, *play* stops playing files and and continues listening for touch tones. However, if the program terminates with a *break* statement, **play** exits immediately. If the *unless* program terminates with *continue,* **Play** does not stop playing *files,* as is the usual case when a tone first arrives. If a TCL

variable used in the *unless* program needs to retain its value for use in other parts of the PhoneScript program, then that variable should be defined before the *unless* program is first run.

options Are one or more keyword-value pairs that can be used to override the default behavior of *play*. The option keywords are:

reason Reason is a character string containing one or more of the characters "tfdx", that indicates the conditions under which the *unless* expression is called. The letters "tfdx" stand for touch tones, a file finiesd playing, all files are finished playing, and the time limit expired, respectively.

nofile_ok
By default, play terminates with an error if the first audio file doesn't exist. If *nofile_ok* is set to one, no error is returned.

type_ahead
If set to 0 (the default is 1) touch tones entered by the user, but not yet processed are discarded as play starts.

ignore Contains a list of touch tones that are completely ignored by *play*. when received from the telephone line.

maxtones
indicates the maximum number of tones that can be entered by the user. before **play** will terminate.

hi_water
sets the maximum number of samples of audio data that have been "played", but have not yet arrived at the telephone interface.

start_tones
Contains a list of tones to initialize the *tones* variable with. Ordinarily, *tones* starts off empty.

timelimit
is the time (in tenths of seconds) to continue accepting touch tones, after all the *files* are done playing, or after the first tone is received.

beep=1 Causes **play** to perform a **beep** command when *files* are done playing.

The variables *tones* and *tone* are set as **play** exits. *Tones* is the return value, whereas *tone* is the last tone returned.

prdate

prdate <format>

prdate <time> <format>

Prdate is used for converting date and time expressions to ASCII strings. With no arguments, *prdate* returns the current time of day as an integer, the number of seconds since January 1st, 1970. This result may be used as the *time* argument to a subsequent call to **prdate,** causing prdate to use *time* instead of the current time and date for its conversion. The *format* argument to **prdate** is an ASCII character string containing special "%X" format codes ala printf, where the "X" represents a letter of the alphabet, and the "%X" is replaced by a time or date expression, depending on the letter used. **Prdate** uses the utility function *strftime()* to do the conversions, and recognizes the same format characters. From the *strftime()* manual they are:

%% same as %

%a day of week, using locale's abbreviated weekday names

%A day of week, using locale's full weekday names

%h month, using locale's abbreviated month names

%B month, using locale’s full month names

%c date and time as %x %X

%C date and time, in locale’s long-format date and time representation

%d day of month (01-31)

%D date as %m/%d/%y

%e day of month (1-31; single digits are preceded by a blank)

%H hour (00-23)

%I hour (00-12)

%j day number of year (001-366) %k hour (0-23; single digits are preceded by a blank)

%l hour (1-12; single digits are preceded by a blank)

%m month number (01-12)

%M minute (00-59)

%n same as

%p locale’s equivalent of AM or PM, whichever is appropriate

%r time as %I:%M:%S %p

%R time as %H:%M

%S seconds (00-59)

%t same as

%T time as %H:%M:%S

%U week number of year (01-52), Sunday is the first day of the week

%w day of week; Sunday is day 0

%W week number of year (01-52), Monday is the first day of the week

%x date, using locale’s date format

%X time, using locale’s time format

%y year within century (00-99)

%Y year, including century (fore example, 1988)

%Z time zone abbreviation

record <file> <list> [record_timeout=<n>] [silence=<n>][truncate=<n>]

The telephone input is digitized and saved in *file* until one of the following conditions is met:

* A tone in the string *list* is received.
* The *record_timeout* TCL variable (or keyword parameter) tenths of seconds has elapsed
* The *silence* TCL variable (or keyword parameter) tenths of seconds has elapsed.

If truncate is indicated, *truncate* bytes are removed from the end of the recorded file after recording stops. This is useful for applications that wish to terminate a recording with a touch tone, without having the tone as part of the recorded message.

sleep <seconds>

The program pauses for *seconds* seconds, with a 100th second granularity.

synth [speed=<n>] [flush=yes]

synth <text>

synth <text> [to] <file>

synth # [to] <file>

The **synth** command invokes Orator™ to convert from ASCII text to speech. The first time **synth** is called, it starts the pipeline of orator processes. With no arguments, the keyword *speed* is used to set the synthesis speed (see the Orator manual for details) if this is the first call to **synth.** With no arguments, **synth** returns the number of synthesized, but not yet played messages. The keyword *flush=yes* will flush any pending messages. The *text* argument is the ASCII text to be synthesized. Currently, *text* MUST end with a period ".". When specified with no *to* argument, the *text* is synthesized by Orator asynchronously, and placed in the orator output queue then finished. Messages in the Orator output Queue are retrieved by using the "#" indicator in a **play** <*files*> parameter, or with the **synth # to ...** command. When the *[to] file* option is specified to **synth, the** *text* (or the message on the Queue if "#" is specified) is copied to *file. File* normally refers to the name of the Unix file to receive the synthesized speech. However, if *file begins with a % ,* **synth** makes up a file name, stores the speech in the made-up name, and places the file name in the TCL variable *file* (with the % removed). All made-up names containing synthesized speech are automatically deleted when the **PhoneScript** program terminates.

EXAMPLES

This is a complete PhoneScript program implements a simple time of day service.

```
# Announce time, date, and fortune (SAU 5/92)

# Initialization, pre-synthesize greeting

set greeting "Hello, the time {9}is."
set rings 2
synth $greeting to %msg
echo "Starting automated time service"

on ringing {
        echo "ringing... " nonl=yes
        }

# Answer a call

on call {
        set time [prdate "%l:%M %p, %A, %B %e, 19-%y."]
        set fortune [exec /usr/games/fortune -s | tr ’´
        synth $time
        synth $fortune
        echo "Call $calls $time... " nonl=yes
        play "$msg # #" until "."
        hangup
        }

on hangup {
        echo "done"
        }

on int {
        echo "Bye, Automated time service signing off"
        }
```

The following program fragment from an answering machine application illustrates the use of the **play**

command. The C style comments are not part of the fragment, but are included for illustrative purposes only.

```
on call {     /* wait for a phone call */
 set messages [glob $msg_dir/{intro,new*,done}.au] /* gather voice messages */
 play $messages until # reason=tf unless {
  case $unless(tone) in {
   "0" {play $help until #          }    /* play help message */
   "1" {incr $(unless(file_pos) -16000}  /* repeat the previous 2 seconds */
   "2" {incr $(unless(file)}             /* go to the next file */
   "3" {incr $(unless(file) -1}          /* go to the previous file */
   "4" {exec "rm -f $unless(file_name)"  /* remove file */
      lreplace $files $file $file       /* take out of file list */
      }
   "5" {set file 1}                     /* go back to the first file */
   ""  {beep}                           /* beep between files */
   }
  continue;                            /* keep playing files */
  }
 hangup
 }
```

FILES

/dev/audio

To read and write digitized audio to the telephone interface.

/dev/call

/dev/answer

The telephone interface for outgoing and incoming calls, respectively.

SEE ALSO

strftime(3) stim(4) Tcl(1)

DIAGNOSTICS

If an error occurs during the execution of the PhoneScript program, a diagnostic message is printed detailing the cause of the problem. The diagnostics will get better in future versions.

The TCL variable *debug* can be set to one or more ASCII characters. Each character causes diagnostic information to be printed about some portion of PhoneScript. The initial value of *debug* is taken from the *DEBUG_LIST* environment variable. At this time the diagnostic characters are not well defined, but some of them are:

"T" Print a message any time a PhoneScript function is started.

"I" print out one time initialization stuff. "p" and "P" Describe the operation of the **play** command.

"s" Describe the operation of the **synth** command.

ENVIRONMENT

PhoneScript looks at several environment variables to potentially override some built in default settings. They are:

RECORD_GAIN

Set the Sparc audio record gain (0-255). This needs to be tuned once for a particular STIM/SPARC combination, then left alone.

PLAY_GAIN

Set the Sparc audio play gain (0-255). This needs to be tuned once for a particular STIM/SPARC combination, then left alone.

QUIET_THRESHOLD
: This is a mu-law value that represents the loudest "quiet" sound. PhoneScript uses this value to determine when a called party has answered. When enough sound samples are louder than this, the call in "answered". (Larger numbers represent quieter sounds.)

SLOW_DIAL
: If set, dialing with the *call* command will be done slowly. This is not normally needed.

GAIN

DRAIN These parameters control the threshold detectors for the call progress monitoring routines. As long as the Sparc record level is set properly, these shouldn't be changed. These parameters will go away when a working automatic gain control section is added to the call progress monitoring code.

BUGS

This is a preliminary version of the manual, describing a preliminary version of PhoneScript. There are bound to be lots of bugs, in the manual and in the implementation. In addition there are many feature that will be added or changed in future versions, both in and not-in the manual.

* on signal should pick up the USR signal any time, not just when waiting for a call
* Time values should be expressed in uniform units.
* The argument parsing in play needs help.
* There might be problems using the TCL time command with *play* or *record.*

AUTHOR

Stephen A. Uhler