Contents

Overview

The CISCO-TELEPRESENCE-CALL-MIB is available in CTS codecs. The MIB provides both signaling and media statistics for the current and past calls on the individual system. A remote network management system can query the CTS codec to download this information proactively. The MIB is somewhat complex in terms of how the table indexing is implemented.

This script is very similar to the Net-SNMP **snmptable** command in that it will walk through the MIB objects to build a column and row format of the content. Instead of displaying the results to stdout, the output will be written to three CSV files. One of which will be a summary of audio/video statistics.

This script should be used only as an example to illustrate **how-to** use the CISCO-TELEPRESENCE-CALL-MIB to download call detailed records, including media statistics.

Requirements

This script is written in BASH and AWK using standard SED and Net-SNMP commands (snmpbulkwalk, snmptranslate, and snmpset).

The following MIBS are required to be installed in your Net-SNMP mibs directory before running the script:

- CISCO-TELEPRESENCE-CALL-MIB
- CISCO-TC
- CISOC-SMI

Tested on RHEL 5+, Fedora 11+, and Mac OS X 10.6+

The script is platform independent and should run on any unix based platform.

Cisco TelePresence Call MIB Example Script

This script will walk the CTS codec in an efficient fashion by using bulkwalk instead of hundreds of single object queries. The system is walked only once to gather the initial set of data. The data is cached and used by the script to build **three** output files. There are over 50+ objects per call detail record. In effort to make the output more human readable, two additional files will be generated in addition to the RAW dump (details) file. The below table details the CDR output types that will be generated.

CDR Output File Types

Type	Description
detail	Verbose -- Provides all objects in row and column format (per codec, per video, per audio, etc.).
brief	A shortened version of the detail, where it only shows key metric values such as jitter, lost and late packets.
summary	Summary will contain all fields as the detail but all stream related fields will be summarized to audio or video.

The script supports command line options for configuration. Below is a list of the command line options available:

Note: Run the script using **cts_callstats.sh <options>**

Required Options

Option	Option Value	Description
-i	<ip or host>	REQUIRED: IP or hostname for CTS
-sp	<password>	REQUIRED: SNMP password (version 3)

Optional Options

Option	Option Value	Description
-h		Help
-r		Reset CTS History Table (reset will happen after record collection)
-sz	<0..500>	Set CTS history table to this size (size will be set after collection and reset)
-o	<file prefix>	Output filename prefix - default is /tmp/cdr-out.<type>.csv (<type> is detail, summary, brief)

Steps to install the script

1. Cut and paste the below script into a text file named **cts_callstats.sh**
2. Edit the first line of the script and update the "**#!/bin/bash**" command to point to your installed **bash** binary
3. (Unix,Linux,Mac OS only) change the mode so that the script will run by using "**chmod 755 cts_callstats.sh**"
4. Optional - If you your CTS snmp verison 3 username is not 'admin' update the **SNMP_USERNAME** variable to indicate the correct username.

5. Load the MIBS in to Net-SNMP directory (normally this is /usr/share/snmp/mibs or /usr/local/share/snmp/mibs)
    - Check which directory your system uses by running: ls -d /usr/share/snmp/mibs ; ls -d /usr/local/share/snmp/mibs
    - FTP the MIB FILES to your machine:
        1. CISCO-TELEPRESENCE-CALL-MIB
        2. CISCO-TC
        3. CISOC-SMI
6. copy the MIBS to your Net-SNMP directory : cp CISCO-*.my /usr/share/snmp/mibs/
7. Run the script by using "**./cts_callstats.sh <options>**"

Script source

```
#!/bin/bash
################################################################################
# cts_callstats.sh
#
# Version: 1.0
#
# Copyright (c) 2009 by Cisco Systems, Inc.
# All rights reserved.
#
# Author : Tim Evens (tievens@cisco.com)
#
# Description :
#
# This script will query the CISCO-TELEPRESENCE-CALL-MIB for call history
#   items and will output CSV row formatted table for call records.
#
# DISCLAIMER:
#   This script is to be used only as an example for how to interact with the
#   CISCO-TELEPRESENCE-CALL-MIB.  Cisco and the author are not
#   responsible for support, any issues, or feature enhancments.  Use at your
#   own risk.
#
#
#
# HISTORY:
#   Jan 28th, 2009 - Tim Evens : Created script
#   Apr 19th, 2011 - Tim Evens : Tested scirpt against CTS-1000 version 1.7.1(4864)
#
################################################################################
shopt -s extglob

# SNMP Username - Normally this is always admin for version 3
SNMP_USERNAME="admin"

# expected programs that this script will use
AWK=awk
SED=sed
SBW=snmpbulkwalk
ST=snmptranslate
SSET=snmpset

# Global Defines
#
# Below defines the call mib objects we are interested in.  The detailed CSV output will
#   be in the order these are listed.  The DETAIL CSV will be the following:
#   CALLMIB_SYSTEM,CALLMIB_INFO,CALLMIB_LATENCY,CALLMIB_STREAM

# ** indexed by 0 or 1
```

```
export CALLMIB_SYSTEM="ctpcLocalAddr ctpcLocalDirNum ctpcMgmtSysAddr";

# ** indexed by <historyIndex>
export CALLMIB_INFO="ctpcRemoteDirNum ctpcLocalSIPCallId ctpcTxDestAddr \
                ctpcStartDateAndTime ctpcDuration ctpcType ctpcSecurity ctpcDirection \
                ctpcState ctpcInitialBitRate ctpcLatestBitRate";

# ** indexed by <historyIndex>.<video|audio>
export CALLMIB_LATENCY="ctpcAvgCallLatency ctpcMaxCallLatency ctpcMaxCallLatencyRecTime";

# ** indexed by <historyIndex>.<video|audio>.<priCodec|auxiliary|...>
export CALLMIB_STREAM="ctpcTxTotalBytes ctpcTxTotalPackets ctpcTxLostPackets ctpcTxCallLostPackets
                  ctpcRxTotalBytes ctpcRxTotalPackets ctpcRxLostPackets ctpcRxCallLostPackets  \
                  ctpcRxOutOfOrderPackets ctpcRxDuplicatePackets ctpcRxLatePackets ctpcRxCallAuth
                  ctpcAvgCallJitter ctpcMaxCallJitter ctpcMaxCallJitterRecTime";

# Below defines what is included in the brief export
export BRIEF_HEADER="${CALLMIB_SYSTEM} histIdx ${CALLMIB_INFO} ctpcAvgCallJitter ctpcMaxCallJitter
                    ctpcMaxCallJitterRecTime ctpcTxLostPackets ctpcRxLostPackets \
                    ctpcRxLatePackets";

# Below are the fields that should be stripped from the summary.  All others from DETAIL will be i
export SUMMARY_FILTER="ctpcLocalSIPCallId ctpcInitialBitRate ${CALLMIB_LATENCY} \
                      ctpcTxCallLostPackets ctpcRxCallLostPackets ctpcMaxCallJitterRecTime";

# --DO NOT CHANGE -- SNMP Options
MIBLOAD="-m CISCO-TELEPRESENCE-CALL-MIB";
SOPTS="${MIBLOAD} -t 5 -r 10 -Lo -OQs -Pe -v 3 -l authnopriv -a md5 -u ${SNMP_USERNAME} -A ";

SOID="1.3.6.1.4.1.9.9.644";
TABLESIZE_OID="$SOID.1.4.6.0";

# Files
TMP_FILE="/tmp/cts-cdr.$$";
CDR_FILE_PREFIX="/tmp/cdr-out";
export CDR_DETAIL_FILE="${CDR_FILE_PREFIX}.detail.csv"
export CDR_SUMMARY_FILE="${CDR_FILE_PREFIX}.summary.csv"
export CDR_BRIEF_FILE="${CDR_FILE_PREFIX}.brief.csv"

#-------------------------------------------------------------------------
# Usage
#    prints the usage for this program
#-------------------------------------------------------------------------
Usage() {
   echo "Usage: $0 <options>";
   echo "  OPTIONS:";
   echo "    -sp <pwd>         = REQUIRED: SNMP passsword (version 3)"
   echo "    -i <ip|host>      = REQUIRED: IP or hostname for CTS"
   echo " "
   echo "    -r                = Reset CTS History Table "
   echo "                            (reset will happen after record collection)"
   echo "    -sz <0..500>      = Set CTS history table to this size"
   echo "                            (size will be set after collection and reset)"
   echo "    -o <file prefix>  = Output filename - default is $CDR_FILE_PREFIX.<type>.csv"
   echo "                    <type> is detail, summary, brief. Three files will be generated.";
   echo "    -h                = Shows help"
}

#-------------------------------------------------------------------------
# Reset_TableSize()
#   This function will reset the CTS CALL MIB history (tablesize)
#
# ENV:
#   uses HOST, SPWD, and TABLESIZE_OID
```

```
#-------------------------------------------------------------------------
Reset_TableSize() {

   echo "RESETTING HISTORY TABLE";

   # Get the old size first
   OLD_SIZE=$($SBW $SOPTS $SPWD $HOST $TABLESIZE_OID | $SED 's/\ [^\ ]\+://g' | $AWK -F "=" '{ pri

   echo "Restore to tablesize: $OLD_SIZE";

   $SSET $SOPTS $SPWD $HOST $TABLESIZE_OID i 0 2> /dev/null > /dev/null;

   # if active call, we can't set to zero, so lets try 1 and 2 to be on the safe side.
   $SSET $SOPTS $SPWD $HOST $TABLESIZE_OID i 1 2> /dev/null > /dev/null;
   $SSET $SOPTS $SPWD $HOST $TABLESIZE_OID i 2 2> /dev/null > /dev/null;
   echo "DEBUG: History Size"
   echo "$SSET $SOPTS $SPWD $HOST $TABLESIZE_OID i 1 2> /dev/null > /dev/null";
   echo "$SSET $SOPTS $SPWD $HOST $TABLESIZE_OID i 2 2> /dev/null > /dev/null";

   # Now set back to original size
   $SSET $SOPTS $SPWD $HOST $TABLESIZE_OID i $OLD_SIZE > /dev/null;
   echo "$SSET $SOPTS $SPWD $HOST $TABLESIZE_OID i $OLD_SIZE > /dev/null";
   if [ $? -ne 0 ]; then
     echo "ERROR: Couldn't set the TableSize back to original value of $OLD_SIZE.";
     echo "       Please check the CTS and MIB settings.";
     exit 20;
   fi
}


#-------------------------------------------------------------------------
# Set_TableSize()
#    This function will set the CTS CALL MIB table size
#
# ARGS:
#   tbsz = table size
#
# ENV:
#   uses HOST, SPWD, and TABLESIZE_OID
#-------------------------------------------------------------------------
Set_TableSize() {
   tbsz=$1;

   echo "SETTING HISTORY TABLE TO $tbsz";

   #  Set the table size
   $SSET $SOPTS $SPWD $HOST $TABLESIZE_OID i $tbsz > /dev/null;
   if [ $? -ne 0 ]; then
     echo "ERROR: Couldn't set the TableSize to value of $tbsz.";
     echo "       Please check the CTS and MIB settings.";
     exit 20;
   fi

}

#-------------------------------------------------------------------------
# Check_MIBLoaded()
#   This function will check if the MIB is loaded correctly
#-------------------------------------------------------------------------
Check_MIBLoaded() {

   if [ "$($ST $MIBLOAD $SOID)" == "CISCO-TELEPRESENCE-CALL-MIB::ciscoTelepresenceCallMIB" ]; ther
      return 0;
   else
```

```
      echo "ERROR: CISCO-TELEPRESENCE-CALL-MIB is not loaded correctly for NetSNMP.";
      echo "       Try checking that you have the following required MIBS in the mibdir";
      echo "       search path, normally /usr/share/snmp/mibs or /usr/local/share/snmp/mibs";
      echo " ";
      echo "   REQUIRED MIBS:";
      echo "     CISCO-TELEPRESENCE-CALL-MIB";
      echo " ";
      echo "   REQUIRED BY IMPORT: these are typical for any Cisco MIB";
      echo "     SNMPv2-SMI";
      echo "     SNMPv2-CONF";
      echo "     SNMPv2-TC";
      echo "     SNMP-FRAMEWORK-MIB";
      echo "     INET-ADDRESS-MIB";
      echo "     CISCO-TC";
      echo "     CISCO-SMI";

      return 1;
   fi
}

#-------------------------------------------------------------------------
# Export_Brief()
#    This function will export the brief CSV file using the CDR_DETAIL_FILE.
#    The brief export is a shorten version of the detail, where it only shows
#    key metric values such as jitter, lost and late packets.
# ENV:
#    This function requires the export BRIEF_HEADER and CDR_DETAIL_FILE
#-------------------------------------------------------------------------
Export_Brief() {

#+++ BEGIN AWK CODE ++++++++++++++++++++++++++++++++++++++
$AWK -F"," '
#-----------------------------------------
# functions
#-----------------------------------------

#
# This function will check the string with the interest list.
#
# RETURNS:
#    true if matched
#    false if not matched
#
function StringInList(string, LIST) {
   rval = 0;                                  # Return value  of func

   # create array and make sure we have objects in array.
   if (split(LIST, a, " ") > 0) {

      # Loop through array searching to see if string matches
      for (ai in a) {
         regex = "^" a[ai] ".*";

         if (string ~ regex)
            return 1;
      }
   }

   return rval;
}


#-----------------------------------------
# Begin code - define starting items
```

```
#-----------------------------------------
BEGIN {
    # Load the header fields that we will keep
    HEADER = ENVIRON["BRIEF_HEADER"];

    first = 1;                                  # First row indicator
}

#-----------------------------------------
# Repeat code - read/load data
#-----------------------------------------
{
   # See if we are on the first row, if so its the header
   if (first) {
      first = 0;

      # Loop through the header columns and store interesting ones
      for (x = 1; x <= NF; x++) {

         # check to see if this header column is in our list
         if (StringInList($x, HEADER)) {

            # It is, print it and save index.
            hdr_idx_list = hdr_idx_list " " x;          # Save index number

            if (length(hdr_nam_list) <= 0)
               hdr_nam_list = $x;                       # Save actual header name
            else
               hdr_nam_list = hdr_nam_list "," $x;      # Save actual header name
         }

      } # End of for (header columns)

      # export the brief header
      print hdr_nam_list > ENVIRON["CDR_BRIEF_FILE"];

   } # end of if (first)

   else {
      # print data rows, but only ones in the list.
      n=split(hdr_idx_list, a, " ");

      # Loop through the columns we want to keep
      for (m = 1; m <= n; m++) {
         if (m == 1)
            printf("%s", $a[m]) >> ENVIRON["CDR_BRIEF_FILE"];   # print column
         else
            printf(",%s", $a[m]) >> ENVIRON["CDR_BRIEF_FILE"];  # print column
      }

      printf("\n") >> ENVIRON["CDR_BRIEF_FILE"];                # End the row

   } # End of else [not first line]

}

#-----------------------------------------
# End code - process what we read
#-----------------------------------------
END {

   printf("Exported brief to %s\n", ENVIRON["CDR_BRIEF_FILE"]);

}
```

```
' $CDR_DETAIL_FILE;
#+++ END AWK CODE ++++++++++++++++++++++++++++++++++++++++


} # End of Export_Brief()

#-------------------------------------------------------------------------
# Export_Summary()
#    This function will export the summary CSV file using the CDR_DETAIL_FILE.
#    The file will contain all fields as the detail but all stream related fields
#    will be summarized to audio or video.
#    Fields marked with Avg will be averged.  Fields with Max will be reported
#    as the max of all, and all others will be summarized.
#
# ENV:
#    This function requires the export SUMMARY_FILTER and CDR_DETAIL_FILE and
#    CDR_SUMMARY_FILE
#-------------------------------------------------------------------------
Export_Summary() {

#+++ BEGIN AWK CODE ++++++++++++++++++++++++++++++++++++++++
$AWK -F"," '
#-----------------------------------------
# functions
#-----------------------------------------

#
# This function will check the string with the interest list.
#
# RETURNS:
#    true if matched
#    false if not matched
#
function StringInList(string, LIST) {
   rval = 0;                                    # Return value  of func

   # create array and make sure we have objects in array.
   if (split(LIST, a, " ") > 0) {

      # Loop through array searching to see if string matches
      for (ai in a) {
         regex = "^" a[ai] ".*";

         if (string ~ regex)
            return 1;
      }
   }

   return rval;
}


#-----------------------------------------
# Begin code - define starting items
#-----------------------------------------
BEGIN {
    hdr_filter = ENVIRON["SUMMARY_FILTER"];     # columns to filter

    first = 1;                                  # First row indicator
}

#-----------------------------------------
# Repeat code - read/load data
#-----------------------------------------
```

```
{
   # See if we are on the first row, if so its the header
   if (first) {
      first = 0;

      # Loop through the header columns and store interesting ones
      for (x = 1; x <= NF; x++) {

         # make sure we only print columns that are not filtered
         if (!StringInList($x, hdr_filter)) {

            # allowed, print it and save index.
            hdr_idx_list = hdr_idx_list " " x;          # Save index number

            # see if we found a stream column
            if ($x ~ /video\..+/ || $x ~ /audio\..+/) {
               # Fix up the header to print only summary column name
               hdr = $x;
               gsub(/\.[^\.]+$/, "", hdr);                      # Remove last index

               arithmetic[x] = hdr;                     # mark this field for arithmetic

               # Fix up the header to print only summary column name
               hdr = $x;
               gsub(/\.[^\.]+$/, "", hdr);                      # Remove last index

               # Use new header list for StringInList match
               hdr_list = hdr_nam_list;
               gsub("," , " ", hdr_list);

               if (!StringInList(hdr, hdr_list))          # Make sure its not already in the list
                  hdr_nam_list = hdr_nam_list "," hdr;

            } # End if (stream column)

            else {
               # now save the header name
               if (length(hdr_nam_list) <= 0)
                  hdr_nam_list = $x;                        # Save actual header name
               else
                  hdr_nam_list = hdr_nam_list "," $x;       # Save actual header name
            }
         } # End of if (StringInList)

      } # end of for (header columns)

      # print the header
      printf("%s", hdr_nam_list) > ENVIRON["CDR_SUMMARY_FILE"];

   } # End of if (first)

   else {    # non-header row
      arith_done = "";                                  # reset

      # Loop through the columns
      for (x = 1; x <= NF; x++) {

         # See if this field requires us to do arithmetic.
         if (x in arithmetic)  {
            if (!StringInList(arithmetic[x], arith_done) ) {

               # Add field to done list so we do not do this again on same summary field
               if (length(arith_done) <= 0)
                  arith_done = arithmetic[x];
```

```
                else
                   arith_done = arith_done " " arithmetic[x];

                val_sum = 0;
                val_max = 0;

                # Loop through arithmetic array so that we apply logic to all columns for this grou
                i = 0;
                for (n in arithmetic) {
                   if (arithmetic[n] == arithmetic[x]) { # Do we have a match
                      # split the value
                      split($n, v, " ");

#printf("%s has %d and val %d (%s)\n", arithmetic[x], n, int(v[1]), v[1]);

                      # Sum the values
                      val_sum += int(v[1]);
                      i++;

                         # find max
                      if (v[1] > val_max)
                         val_max = int(v[1]);
                   }
                } # End of for (calc)

                # get avgerage
                val_avg = val_sum / i;

#printf("arithmetric on %d-%s (%d/%d/%d)\n", x, arithmetic[x], val_sum, val_max, val_avg);

                # Get the value desc
                split($x, v, " ");

                # Print the value needed
                if (arithmetic[x] ~ /Avg/)
                   printf(",%ld %s", val_avg, v[2]) >> ENVIRON["CDR_SUMMARY_FILE"];
                else if (arithmetic[x] ~ /Max/)
                   printf(",%ld %s", val_max, v[2]) >> ENVIRON["CDR_SUMMARY_FILE"];
                else
                   printf(",%ld %s", val_sum, v[2]) >> ENVIRON["CDR_SUMMARY_FILE"];
             } # End of if (StringInList(not already done))
          } # End of if (arithmetic)

          # make sure to only print the columns that were not filtered
          else {
             split(hdr_idx_list, a, " ");
             for (n in a) {
                if (a[n] == x) {
                   if (x == 1)
                      printf("%s", $x) >> ENVIRON["CDR_SUMMARY_FILE"];
                   else
                      printf(",%s", $x) >> ENVIRON["CDR_SUMMARY_FILE"];

                   break;
                }
             }
          }
       } # end of for (columns)
    } # end of else [ non-header row ]

    # Add newline to end record
    printf("\n") >> ENVIRON["CDR_SUMMARY_FILE"];
}
```

```
#-----------------------------------------
# End code - process what we read
#-----------------------------------------
END {

   printf("Exported summary to %s\n", ENVIRON["CDR_SUMMARY_FILE"]);

}

' $CDR_DETAIL_FILE;
#+++ END AWK CODE ++++++++++++++++++++++++++++++++++++++

} # End of Export_Summary()


#-------------------------------------------------------------------------------
# Get_CDR()
#   Walk the MIB, parse it and output the call records in CSV format.
#      This function includes the AWK code for detailed record export.
#      This is here because it's required for the summary and breif exports.
#
# ENV:
#   This fucntion does not take args, but it does expect the TMP_FILE,
#     HOST, SPW, and CDR_DETAIL_FILE ENV's are set.
#-------------------------------------------------------------------------------
Get_CDR() {

   echo "GETTING SNMP DATA FROM CTS";

   # We first need to get the data into a local file so that we can work with it.
   $SBW $SOPTS $SPWD $HOST $SOID > $TMP_FILE;
   echo "DEBUG: Pull Command";
   echo "$SBW $SOPTS $SPWD $HOST $SOID > $TMP_FILE";
   if [ $? -ne 0 ]; then
      echo "ERROR: SNMP failed.  Check host IP and snmppassword and ensure SNMP is running on CTS.
      echo "---DEBUG - 6 lines from $TMP_FILE ----------";
      head -6 $TMP_FILE
      echo "---DEBUG END--------------------------------";
      exit 10;
   fi

   # Make sure we have some data in the tmp file
   CHK=$(wc -l $TMP_FILE | $AWK '{print $1}' );
   if [ $CHK -lt 5 ]; then
      echo "ERROR: SNMP failed to collect CDR information.  Check that SNMP is running on CTS.";
      echo "---DEBUG - 6 lines from $TMP_FILE ----------";
      head -6 $TMP_FILE
      echo "---DEBUG END--------------------------------";
      exit 11;
   fi
   # Some snmp versions have problems with -OQ, and will print INTEGER/Counter32 etc..
   #   Let's strip that off
   #mv ${TMP_FILE} ${TMP_FILE}.1
   #$SED 's/\ [^\ ]\+://g' ${TMP_FILE}.1 > $TMP_FILE;
   #rm -f ${TMP_FILE}.1

   echo "PARSING DATA";

   # Parse and build the primary CDR detailed record file.
#+++ BEGIN AWK CODE ++++++++++++++++++++++++++++++++++++
#--- We use AWK since it performs better than running a ton of egrep/sed commands
#--- for every line in the snmp results file.
$AWK -F' = ' '
```

```
#-----------------------------------------
# functions
#-----------------------------------------

#
# This function will remove unwanted characters in the string
# RETURNS: returns the updated string
#
function Strip(string) {

   gsub(/"/, "", string);
   gsub(/,/, " ", string);
   gsub(/\|/, " ", string);

   return string;
}

#
# This function will check the oidnam with the interest list.
#
# RETURNS:
#    true if matched
#    false if not matched
#
function CheckOIDName(oidnam, MIBOBJS) {
   rval = 0;                                    # Return value  of func

   # create array and make sure we have objects in array.
   if (split(MIBOBJS, a, " ") > 0) {

      # Loop through array searching to see if oidnam matches
      for (ai in a) {
         #if (match(toupper(a[ai]), toupper(oidnam)) )
         if (toupper(a[ai]) == toupper(oidnam) )
            return 1;
      }
   }

   return rval;
}

#
# This function will get the digit index from the oidnam
# RETURNS: returns HistoryIndex
#
function Get_histIdx(oidnam) {
   split(oidnam, a, ".");
   return a[2];
}

#
# This function will get the oid name from the oid var
# RETURNS: returns oidnam
#
function Get_OidNam(oidnam) {
  split(oidnam, a, ".");
  return a[1];
}

#-----------------------------------------
# Begin code - define starting items
#-----------------------------------------
BEGIN {
   # Global Defines
```

```
   #
   # Below defines the call mib objects we are interested in.  The CSV output will
   #   be in the order these are listed in these arrays.  The CSV will be the following:
   #   CALLMIB_SYSTEM,CALLMIB_INFO,CALLMIB_LATENCY,CALLMIB_STREAM
   #
   # To make it easier for the program user, these are defined outside of AWK in the main BASH scr

   # ** indexed by 0 or 1
   CALLMIB_SYSTEM = ENVIRON["CALLMIB_SYSTEM"];

   # ** indexed by <historyIndex>
   CALLMIB_INFO = ENVIRON["CALLMIB_INFO"];

   # ** indexed by <historyIndex>.<video|audio>
   CALLMIB_LATENCY = ENVIRON["CALLMIB_LATENCY"];

   # ** indexed by <historyIndex>.<video|audio>.<priCodec|auxiliary|...>
   CALLMIB_STREAM = ENVIRON["CALLMIB_STREAM"];

   # Global vars
   i = 0;
   localaddr = "";
   localdn = "";
   mgmtaddr = "";
   stream_idx_list = "";                        # Stream indexes for cdr records
}

#-----------------------------------------
# Repeat code - read/load data
#-----------------------------------------
{
   oidnam = Get_OidNam($1);                     # Save the OID name, strip the index and other ite
   histIdx = Get_histIdx($1);                   # Save the history idx

   # Check if we have a single index
   if (/^[^\.]+\.[^\.]+ = / ) {

      # Check if this is general system items
      if (CheckOIDName(oidnam, CALLMIB_SYSTEM)) {

         # Store the value
         if (oidnam ~ /LocalAddr/)
            localaddr = Strip($2);
         else if (oidnam ~ /LocalDirNum/)
            localdn = Strip($2);
         else if (oidnam ~ /MgmtSysAddr/)
            mgmtaddr = Strip($2);
      }

      # Check if this is a call info item, if so store it.
      else if (CheckOIDName(oidnam, CALLMIB_INFO)) {

         # If first record, add history index
         if (length(cdr[histIdx]) <= 0)
            cdr[histIdx] = "histIdx=" histIdx;

         # store in cdr array
         cdr[histIdx] = cdr[histIdx] "|" oidnam "=" Strip($2);
      }
   }

   # check for double index
   else if (/^[^\.]+\.[^\.]+\.[^\.]+ = / ) {
      if (CheckOIDName(oidnam, CALLMIB_LATENCY)) {
```

```
         # Fix oid to strip out histIdx
         oidnam = $1;
         gsub(/\.[0-9]+/, "", oidnam);

         # store in cdr array
         cdr[histIdx] = cdr[histIdx] "|" oidnam "=" Strip($2);
      }
   }

   # check for triple index
   else if (/^[^\.]+\.[^\.]+\.[^\.]+\.[^\.]+ = / ) {
      if (CheckOIDName(oidnam, CALLMIB_STREAM)) {
         # Fix oid to strip out histIdx
         oidnam = $1;
         gsub(/\.[0-9]+/, "", oidnam);

         # Store the index
         stream_idx = oidnam;
         gsub(/^[^\.]+\./, "", stream_idx);

         # Before storing this one, check if we already have it in the list
         if (!CheckOIDName(stream_idx, stream_idx_list))
            # Save new index to index list
            stream_idx_list = stream_idx_list " " stream_idx;

         # store in cdr array
         cdr[histIdx] = cdr[histIdx] "|" oidnam "=" Strip($2);
      }
   }

   ++i;
}


#-----------------------------------------
# End code - process what we read
#-----------------------------------------
END {
   if (!length(localaddr) || !length(localdn)) {
      printf("ERROR: Failed to parse SNMP CISCO-TELEPRESENCE-CALL-MIB walk.  Check\n");
      printf("       syntax of SNMPWALK and ensure that the CISCO-TELEPRESENCE-CALL-MIB is loaded!
      exit 1;
   }

   # print user status of what we are doing now.
   printf("Read %d lines, now exporting CDR detail to %s\n", i, ENVIRON["CDR_DETAIL_FILE"]);

   # build the header line for CSV file
   header = "ctpcLocalAddr,ctpcLocalDirNum,ctpcMgmtSysAddr,histIdx";

   m=split(CALLMIB_INFO,a, " ");
   for (n = 1; n <= m; n++)
      header = header "," a[n];

   m=split(CALLMIB_LATENCY, a, " ");
   for (n = 1; n <= m; n++)
      header = header "," a[n] ".audio";
   for (n = 1; n <= m; n++)
      header = header "," a[n] ".video";

   m=split(CALLMIB_STREAM, a, " ");
   for (n = 1; n <= m; n++) {

      # This gets a bit tricky...  For each in this array list we can have
```

```
      #   variable stream types/sources, thus we need to append the index per object.
      x=split(stream_idx_list, a2, " ");
      for (y = 1; y <= x; y++) {
        header = header "," a[n] "." a2[y];
      }
   }

   # Print the header
   print header > ENVIRON["CDR_DETAIL_FILE"];

   # Print out the CDR records, but in the order that the header is written.
   for (ai in cdr) {                          # Loop through cdr records
      # make array from header
      m=split(header, a, ",");                # a is array of header columns

      # make array from cdr
      x=split(cdr[ai], a2, "|");             # a2 is array of cdr record columns

      # Provide output for user to know what we are doing.
      printf("Processing %s...\n", a2[1]);   # Provide status

      # Loop through each header item and print assoc column from cdr
      for (n = 4; n <= m; n++) {

         # Now find the column that should be printed from cdr record
         #for (y = 1; y <= x; y++) {          # Loop through each column in cdr
         found = 0;
         for (y in a2) {                      # Loop through each column in cdr
            # Check the column is a match for header column
            split(a2[y], varval, "=");
            if (toupper(a[n]) == toupper(varval[1]) ) {
#            if (match(toupper(a[n]), toupper(varval[1])) ) {

               # If the 4th object, append system items after histIdx.
               if (n == 4)
                  printf("%s,%s,%s", localaddr, localdn, mgmtaddr) >> ENVIRON["CDR_DETAIL_FILE"];

               printf(",%s", varval[2]) >> ENVIRON["CDR_DETAIL_FILE"];  # Found match, print it

               found = 1;
               delete a2[y];                  # delete this object as we are done with it.
               break;
            }
         } # end of for (cdr columns)

         # Check if header column was printed, if not print a blank one as it must be missing.
         if (!found)
            printf(",") >> ENVIRON["CDR_DETAIL_FILE"];

      } # end of for (header items)

      # print newline
      printf ("\n") >> ENVIRON["CDR_DETAIL_FILE"];      # indicate next row in csv

   } # end of for (cdr records)

#   for (ai in cdr) {
#      printf("CDR[%d] = %s\n", ai, cdr[ai]);
#   }
}

' $TMP_FILE;

#+++ END AWK CODE ++++++++++++++++++++++++++++++++++++++++
```

```
  # Export the other reports now that we have the detailed one
  Export_Brief;
  Export_Summary;

  # Remove temp file
  rm -f $TMP_FILE;

} # end of Get_CDR()


#-------------------------------------------------------------------------------
# main
#-------------------------------------------------------------------------------
   TB_SZ=-2                           # indicates to not set table size by default
   TB_RESET=0                         # indicates to not reset the table history

   # Set the trap hander to remove temp file
   trap 'rm -f $TMP_FILE; exit $?' KILL INT TERM EXIT

   # Check that the MIB is loaded correctly.
   Check_MIBLoaded
   if [ $? -gt 0 ]; then
      exit 1;
   fi

   # Check command line args
   if [ $# -lt 4 ]; then
      echo "ERROR: You need at least 4 args, try again."
      Usage
      exit 1;

   else # proceed to check args
      while [ $1 ]; do
         # Check check args
         case $1 in
            -h) # Help
                Usage
                exit 0;
               ;;

            -sp) # SNMP password
                 shift
                 SPWD=$1
               ;;

            -i) # IP/Host of CTS
                shift
                HOST=$1
               ;;

            -o) # Output file prefix
                shift
                CDR_FILE_PREFIX=$1;
                export CDR_DETAIL_FILE="${CDR_FILE_PREFIX}.detail.csv"
                export CDR_SUMMARY_FILE="${CDR_FILE_PREFIX}.summary.csv"
                export CDR_BRIEF_FILE="${CDR_FILE_PREFIX}.brief.csv"
               ;;

            -r) # Reset history/tablesize
               TB_RESET=1;
               ;;

            -sz) # set table size
```

```
                shift

                TB_SZ=$1;
                ;;

             *) # Default is to use oldMain
                echo "ERROR: Invalid arg '$1'";
                Usage;
                exit 2;
                ;;
          esac

          shift;                          # Move up an arg
       done

       # Now that we finished parsing the args, lets make sure we have what we need.
       if [ -z ${SPWD:=""} -o -z ${HOST:=""}  ]; then
          echo "ERROR: Oh no, you forgot to supply the IP address/host and/or SNMP password."
          Usage;
          exit 2;
       fi

    fi # End of checking args


    # Got what we need now, lets do the work.
    Get_CDR;

    # If we need to reset the history/tablesize lets do it now
    if [ ${TB_RESET} -gt 0 ]; then
       Reset_TableSize;
    fi

    # If we need to set the table size to a new value lets do that now
    if [ ${TB_SZ} -ge 0 ];  then
       Set_TableSize $TB_SZ;
    fi

    echo "Done";

# END
```

Examples

Retrieving call records from a CTS-3000

```
mymachine:scripts tievens$ ./cts_callstats.sh  -sp snmppassword -i 10.22.138.131 -sz 100 -o /tmp/T
GETTING SNMP DATA FROM CTS
DEBUG: Pull Command
snmpbulkwalk -m CISCO-TELEPRESENCE-CALL-MIB -t 5 -r 10 -Lo -OQs -Pe -v 3 -l authnopriv -a md5 -u a
PARSING DATA
Read 652 lines, now exporting CDR detail to /tmp/Tim-CTS-3k_cdr.detail.csv
Processing histIdx=2...
Processing histIdx=1...
Exported brief to /tmp/Tim-CTS-3k_cdr.brief.csv
Exported summary to /tmp/Tim-CTS-3k_cdr.summary.csv
SETTING HISTORY TABLE TO 100
Done
```

Brief Output

```
ctpcLocalAddr,ctpcLocalDirNum,ctpcMgmtSysAddr,histIdx,ctpcRemoteDirNum,ctpcLocalSIPCallId,ctpcTxDe
10.22.138.131,14166140001,10.22.138.132,2,19255290001,1d9f900-dad1b163-36-848a160a@10.22.138.132,1
10.22.138.131,14166140001,10.22.138.132,1,14166140002,9b54db00-dad1b0b7-1e-848a160a@10.22.138.132,
```

Summary Output

```
ctpcLocalAddr,ctpcLocalDirNum,ctpcMgmtSysAddr,histIdx,ctpcRemoteDirNum,ctpcTxDestAddr,ctpcStartDat
10.22.138.131,14166140001,10.22.138.132,2,19255290001,10.22.139.204,2011-4-19 15:59:34.0 -0:0,13 s
10.22.138.131,14166140001,10.22.138.132,1,14166140002,10.22.139.204,2011-4-19 15:56:48.0 -0:0,34 s
```

Detail Output

```
ctpcLocalAddr,ctpcLocalDirNum,ctpcMgmtSysAddr,histIdx,ctpcRemoteDirNum,ctpcLocalSIPCallId,ctpcTxDe
10.22.138.131,14166140001,10.22.138.132,2,19255290001,1d9f900-dad1b163-36-848a160a@10.22.138.132,1
10.22.138.131,14166140001,10.22.138.132,1,14166140002,9b54db00-dad1b0b7-1e-848a160a@10.22.138.132,
```