Time-Zone Processing with Asterisk, Part I

Matthew Gast

Abstract

Use Asterisk to adjust for your daylight hours.

Last year, I took a trip to Asia. To stay in touch, I carried a GSM world phone, capable of receiving telephone calls in the countries I was visiting. The capability to receive calls with the same mobile phone number I use at home while halfway across the world seemed incredibly cool—at least until the first call came in! Mobile phones hide the location of the phone, which cuts both ways. A colleague had decided to call me in the middle of the day on a Friday, which had awakened me very early on Saturday morning, because the phone “hid” my faraway location from him.

After returning home, I asked several people why my phone company could not simply play a message to warn callers when my time zone changes by more than four or five hours, letting them know the call might be inconvenient. Nobody could come up with a technical reason, but we all suspected it was because the mobile phone company to which I subscribed charged several dollars per minute to connect calls. As part of the process of attaching a GSM phone to a network, the home network needs to learn where the phone is visiting, and that information conceivably could include a time zone.

I returned to my idea once I started using Asterisk, because it provides an extensive toolkit for designing PBX-hosted services. Anything that can be coded in a computer can become an Asterisk service. After I understood the basics of Asterisk, I sat down to implement a feature that kept track of the time of day where I visited and prevented calls from coming in at inconvenient times.

The system I built on top of Asterisk to handle this feature has two major parts. The key to the system is maintaining a time-zone offset from the time in London. (My code implements offsets only of whole hours, though it could be extended to use either half or quarter hours.) When a device first connects to Asterisk, its IP address is used to guess the location and, therefore, the time offset. After the offset is programmed into the system, incoming calls are then checked against the time at the remote location. Before the phone is allowed to ring, the time at the remote location is checked, and callers can be warned if they are trying to complete a call at an inconvenient time.

Step 1: Estimating the Time Zone

Asterisk does not have a built-in method to estimate the time zone from an IP address, but it does have the next best thing—the Asterisk Gateway Interface (AGI). AGI programming allows an Asterisk extension to pass data to an external program, do computations in that program and return results as Asterisk channel variables.

I began the project by writing an AGI script that would take an IP address as input and return an estimated time zone. Several existing geolocation databases map IP addresses into geographic information. None of the free products or compilations I tried for this project could return a time zone directly from the database, so I estimated the time zone based on the longitude. (The earth's surface has...
24 time zones, each of which is approximately 15 degrees of longitude.) After trying several databases, I settled on MaxMind's GeoLite City, a free-for-non-commercial-use version of the comprehensive GeoCity commercial database.

GeoCity has APIs available in several programming languages. I used Perl, because there is also an AGI library module for Perl that makes handling AGI scripts easier. As input, the program takes an IP address, and then it needs to return the time zone. I used the convention of returning the time zone as an offset of the number of hours from the time in London, which leads to some differences in daylight time handling.

The start of the program pulls in utility functions, including the Asterisk::AGI module, which decodes all the parameters passed to the program by Asterisk:

```perl
#!/usr/bin/perl -w
# Asterisk AGI to estimate time zone from IP address
use strict;
use Asterisk::AGI;
use Geo::IP::PurePerl;
use POSIX qw(ceil floor);

# California is GMT -8
my $HOME_OFFSET = -8;
my $AGI = new Asterisk::AGI;
my %input = $AGI->ReadParse();

(The full listing of programs from this article is available on the Linux Journal FTP site; see Resources.)

The argument to the program is an IP address, which is given to us by Asterisk. The first check is to determine whether the IP address is on the local subnet of the Asterisk server, 192.168.1.0/24. Most location databases don't include RFC 1918 address space and won't return a lookup. The MaxMind API can accept a domain name as an argument, but we don't expect to pass it one:

```perl
my $addr = $ARGV[0];
my @octets = split(/\./,$addr);
if (($octets[0] eq "192") && ($octets[1] eq "168") && ($octets[2] eq "1")) {
    # Local IP addresses get the home offset
    $AGI->set_variable("TZ_OFFSET","$HOME_OFFSET");
    exit 0;
}
```

The use of the database is straightforward. We create a new object, telling the API the location of the database on disk, and then call the get_city_record_as_hash function in MaxMind's API, which returns everything about the IP address as a hash. The item of interest is the longitude component of the hash. If there isn't one, we'll simply return -8 for California and let Asterisk deal with the problem:

```perl
my $gi = Geo::IP::PurePerl-> new(
    "/usr/local/share/GeoIP/GeoIP/GeoLiteCity.dat",
    GEOIP_STANDARD);
my $cityref = $gi->get_city_record_as_hash($addr);
if (!(defined ($cityref->["longitude"]))) {
    # Guess at the home time when longitude undefined
    $AGI->set_variable("TZ_OFFSET",-8);
    exit 1;
}
```
my $longitude=$cityref->{"longitude"};

A bit of math is required to deal with the fact that time-zone boundaries may be 15 degrees, but zero degrees is in the middle of a time zone. We can use two formulas, depending on whether the longitude is positive or negative. After computing the time zone, we pass it back to Asterisk in the TZ_OFFSET channel variable, where it is available for use in the dial plan:

my $numerator;
my $denominator=15;
if ($longitude>0) {
    $numerator=$longitude+7.5;
    $quotient=floor($numerator/$denominator);
} else {
    $numerator=$longitude-7.5;
    $quotient=ceil($numerator/$denominator);
}

$AGI->set_variable("TZ_OFFSET",$quotient);

Step 2: Confirming the Time-Zone Information

As convenient as it would be to have a reliable database of IP addresses mapped into the right time zones, there is still the problem of handling summer shifts in time. Plus, the estimate comes from a demonstration database that is not guaranteed to be accurate. Therefore, the AGI script is called from within an extension that is used to confirm the estimate or save a new one. For the confirmation step, I created an extension with the number *89 (because 8–9 is the numerical mapping of the letters T–Z). Just as with the previous program, some of the debugging statements are removed for brevity, but the full version is available from the LJ FTP site (see Resources).

This article shows the dial plan information entry in the Asterisk Expression Language (AEL). I started using AEL because it has better control structures and it is much easier to write structured code. For voice menus, the superior control structures in AEL allow much easier validation of input.

A call into the extension begins with a welcome announcement and a call to the script (shown previously) to estimate the time zone. All of the time-zone confirmation greetings are stored in the msg/tz subdirectory of the Asterisk sound library. Asterisk's SIPCHANINFO can be used to get SIP channel information. Specifically, the value of SIPCHANINFO(recvip) is the routable Internet address that the remote device used to register and, therefore, will work even if the remote device is behind a network address translator:

_*89 => {
    Answer;
    Playback(msg/tz/tz-wizard-welcome);
    Set(PEERIP=${SIPCHANINFO(recvip)});

    // Geolocate as a first stab at time zone
    AGI(tz-lookup.pl,${PEERIP});
    NoOp(TZ offset from script is ${TZ_OFFSET});
}

The script returns a guess at the time zone of the IP address used by the SIP peer in the TZ_OFFSET variable. However, the whole purpose of this call is to confirm the offset, so we proceed to a series of confirmation steps. The system starts by giving a confirmation of the time offset from the time in London and uses that to say the time. Asterisk keeps internal time as epoch time (the number of seconds
past midnight on January 1, 1970 GMT) and converts it to local time for a given time zone. I am handling any adjustments to the time for summer months with a sledgehammer, which assumes that most of the countries I visit will be on roughly the same schedule as the UK and corrects any errors later in the validation step:

```plaintext
playoffset:
  Playback(msg/tz/tz-you-are-at);
  SayDigits($TZ_OFFSET);
  Playback(msg/tz/tz-hours-to-london);

playtime:
  // London time keeps summer time
  Playback(msg/tz/tz-current-time-is);
  SayUnixTime($[${EPOCH}+${TZ_OFFSET}*60*60],
              Europe/London,'digits/at' IMp);

Next, we ask the user to confirm whether the time is correct. The Read() application gets one digit from the user. AEL's switch statement is very handy for working with user input, because it can be used to set up a series of actions for a voice menu quite easily without extensive use of a forest of branching statements. In this case, the switch statement offers the option of a one-hour correction by entering the number 1, an arbitrary correction with the number 2 and an error statement that jumps back to the start of the time readout if anything else is pressed. The only catch in using goto within a switch statement in AEL is that due to the internal representation of the control structures, you must use a fully qualified goto, including the context, extension number and label. My internal extension context is from-internal, so gotos are prefaced with from-internal|*89:

```plaintext
Read(INPUT,msg/tz/tz-confirm-correct,1);
switch (${INPUT}) {
  case 1:
    goto from-internal|*89|expiry;
  case 2:
    goto from-internal|*89|correction;
  default:
    Playback(msg/tz/tz-1-or-2-please);
    goto from-internal|*89|playoffset;
};
```

At the correction label, there is a second option. I expect that summer-time errors will be common in actual use, so I added an expedited one-hour correction that can add or subtract an hour easily. The menu that handles the type of correction and the one-hour correction subroutine is available on the LJ FTP site. In structure, both are similar to the switch statement shown previously.

When I arrive in faraway locations, I am usually quite tired, and my ability to do mental arithmetic is substantially reduced from my capacity when fully functional. Rather than a voice menu to select a location, the menu prompts for the local clock time and computes the time-zone offset from the time in London. The basic algorithm is to obtain the 24-hour reference time in London with Asterisk's STRFTIME function and compute the offset from the time entered by the user. There's a possibility that the resulting offset will be too big or too small, so the script corrects for that:

```plaintext
gmtskew:
  Read(INPUT,msg/tz/tz-24-hour-prompt,4);
  Set(REMOTEHR=${INPUT:0:${LEN(${INPUT})-2]});
  Set(REFERENCEHR=${STRFTIME(${EPOCH},
                  Europe/London,%H));
  Set(TZ_OFFSET=${REMOTEHR}-${REFERENCEHR});
  if ( ${TZ_OFFSET} > 12 ) {}
Set(TZ_OFFSET=${TZ_OFFSET}-24 );
}
if ( ${TZ_OFFSET} < -12 ) {
    Set(TZ_OFFSET=${TZ_OFFSET}+24 );
}
Return;

When the user confirms the time offset, the changes are saved in the Asterisk database with the DB function. Part of saving the changes is to ask the user for how long the offset should be saved. This code prompts for the number of days, though that part of the code easily can be extended to ask for an expiration date and time. After determining the time when the offset expires, the offset is read back to the caller in both the time at the remote site, as well as the home location. (Note that my home location is US/Pacific; you will need to replace that with your own time zone.)

For record-keeping purposes, four variables are stored with the name of the extension that controls the offset. There is the actual offset, as well as a start time, expiration time and the IP address of the SIP peer. If the device is moved before the end of the offset, we want to re-confirm the time offset automatically:

```plaintext
expiry:
    Set(NOW=${EPOCH});
    Set(CURRENT_OFFSET_TIME=${EPOCH} + ${TZ_OFFSET}*60*60 );
    Set(DB(tz/${PEERNAME}-TIMESKEW)=$[ ${PEERNAME}];
    Set(DB(tz/${PEERNAME}-TIMESKEW_START)=$[ ${NOW}]);
    Set(DB(tz/${PEERNAME}-TIMESKEW_ADDR)=$[PEERIP]);
    Playback(msg/tz/tz-your-offset-of);
    SayDigits($[TZ_OFFSET]);
    Playback(msg/tz/tz-hours-relative-to-london);
    Playback(msg/tz/tz-confirm-time);
    SayUnixTime($[CURRENT_OFFSET_TIME],
        Europe/London,A \"digits/at\" IMP);
expiration-confirm:
    Read(TZ_DURATION,msg/tz/tz-days-active,2);
expiration-readout:
    Set(DB(tz/${PEERNAME}-TIMESKEW_END)=$[NOW]+24*60*60*$[TZ_DURATION]);
    Playback(msg/tz/tz-shift-active-for);
    SayDigits($[TZ_DURATION]);
    Playback(msg/tz/tz-days);
    Read(INPUT,msg/tz/1-if-right--2-if-wrong,1);
    switch ($[INPUT]) {
        case 1:
            // Everything is OK, read out results
            NoOp( Go to result read-out);
            break;
        case 2:
            goto *89|expiration-confirm;
        default:
            Playback(msg/tz/tz-1-or-2-please);
            goto *89|expiration-readout;
    }
```

Step 3: Letting Callers Know the Time

At this point, Asterisk has all the data it needs to restrict calls based on the time of day. The Asterisk dial plan can be used to check the time at the remote site; if it is before, say, 8 am or after 10 pm, the
Time-Zone Processing with Asterisk, Part I

phone plays the remote time to the caller and asks whether the extension should ring anyway.

First, Asterisk needs to pick up the phone and compare the current local time to the 8 am to 10 pm window. Asterisk's STRFTIME function converts an epoch into a time of day. By adjusting the current epoch time with the offset value, the STRFTIME function returns a time of day. A call that is too early or too late jumps to code that silences the ringer. When a call's ring is silenced, this example code allows the caller an override:

```plaintext
300 => {
    Answer;
    Playback(msg/remote-extension-greeting);
    Set(TZ_OFFSET=${DB(tz/${EXTEN}-TIMESKEW)});
    Set(RMT_EPOCH=${EPOCH}+${TZ_OFFSET}*60*60});
    Set(REMOTE_CLOCK=${STRFTIME(${RMT_EPOCH},
        Europe/London,%H:%M)});
    if("${REMOTE_CLOCK}" < "08:00") {
        goto 300|too-early;
    };
    if("${REMOTE_CLOCK}" >= "22:00") {
        goto 300|too-late;
    };
    goto 300|normal-ring;
}
```

The confirmation code starts by silencing the ring, but allows the user to enable ringing by pressing one. I've allowed any caller the flexibility to override my hours, but this conceivably could be handled by allowing only select callers to be able to override. The “standard” way to set the ring tone is to set the ALERT_INFO channel variable. My ATA is a Sipura, which allows the definition of eight ring cadences. I have defined a cadence called silence, which is a ring that never uses the bell:

too-early:
    NoOp(Too early to ring);
too-late:
    NoOp(Too late to ring);
override-silence:
    Playback(msg/my-remote-time-is);
    SayUnixTime(${RMT_EPOCH},Europe/London, A \`digits/a\' IMP);
    Read(CONFIRM,msg/press-1-to-confirm-call,1);
    if (${CONFIRM}=1) {
        Set(__ALERT_INFO=Bellcore-r1);
    };

Ringing the phone is straightforward, because it requires using only the Dial application, and the ring cadence has been set elsewhere. Alternatively, the call can be sent straight to voice mail:

```plaintext
normal-ring:
    Dial(SIP/300,20);
vm-only:
    VoiceMail(umatthew);
Hangup;
}
```

For enhanced code re-use, the time-of-day checks could be incorporated into a macro that is called as...
part of every extension.

Using This Project

As a first step, install the GeoLiteCity database from MaxMind, and install the time-zone lookup script into /var/lib/asterisk/agi-bin. To call that script, add the time-zone configuration extension *89 to your dial plan. Every extension that requires time-of-day-specific treatment needs to have its dial plan modified with code similar to that shown in Step 3 of this article.

Then, as a user of the PBX, every time a SIP extension is registered, you need to call *89 to set up the time zone. This need to initiate the configuration process manually is somewhat annoying. Asterisk does provide an interface that can be used to set up the call to the user automatically, which I will describe in a follow-up article.

On a PBX that supports multiple users, several items would benefit from centralized storage. In this example, the time-of-day comparison is coded into the target extension's dial plan. By storing this data in the Asterisk database, it would be possible to let users change their time-of-day schedules without administrator intervention.

Finally, I also have configured my PBX with a “friends and family” override feature that allows selected callers to complete a call even if it normally would be blocked. Callers on the privileged list can request the time of day at my location and are allowed to ring a call through even if it would ordinarily be blocked.

Resources 9190s1.qrk

Full source listings of all files: ftp://ftp.linuxjournal.com/pub/lj/issue155/9190.tgz (The dial plan code for the time-zone confirmation menu is 9190l1.txt, the dial plan code for an extension that is time-zone-aware is 9190l2.txt and the AGI script for time-zone geolocation lookup is 9190l3.txt.)

MaxMind GeoLite City Database (Download and API): http://www.maxmind.com/app/geolitecity

Asterisk: http://www.asterisk.org

Asterisk AEL: http://www.voip-info.org/wiki/view/Asterisk+AEL