Build your own calendar alerting system

Meeting Time!

A Perl daemon reads iCalendar files with meeting dates and alerts the user before the meeting is due to start. *By Michael Schilli*

People who stubbornly refuse to use Microsoft Exchange to handle communications at work and prefer to avoid GUI-laden calendar applications are likely to receive invitations to meetings by email in the form of `.ics` files.

These machine-readable text files in iCalendar format [2] describe the date and time of the meeting, the subject for discussion, and who else will be attending. They also define the cycle for meetings that recur regularly at the same time every day or on the same day every week.

GUI-Laden or Perl-Light?

Calendar applications from Gnome and KDE, Evolution, iCal on the Mac, Outlook on Windows, or Google Calendar on the web can all import `.ics` files and display meetings in a colorful overview (Figure 1). They also pop up dialog boxes that alert users to imminent meeting appointments and send them scuttling down the corridor to the conference room.

Programs such as Google Calendar also let you export your calendar data to `.ics` files. This possibility opens up the door to do-it-yourself calendar programs, such as the ical-daemon I will be looking at in this article. This daemon parses a series of `.ics` files, creates an alert table with the imminent meetings, and executes a script (`ical-notify`) 15 minutes before a meeting is due to start to wake up the user in any way you see fit.

Email is possible, but it could just as easily be a message on an IM or IRC network – or something completely different, such as playing a particular music track.

Exporting your Calendar

To download calendar data from the Google server, you need to click the *Export* button in your Google calendar below *Settings* | *Google Calendar Settings* | *Calendars*. Doing this gives you a ZIP archive with a `.ics` file (see Figures 2 and 3).

If you take a close look at the `.ics` file in Figure 4, you will see lines of tags, in which `DTSTART` indicates the start of a meeting and `DESCRIPTION` provides the topic for discussion. This is a 1:1 meeting with my manager that takes place on Wednesday every other week, as defined by this line:

MIKE SCHILLI

Mike Schilli works as a software engineer with Yahoo! in Sunnyvale, California. He can be contacted at *mschilli@perlmeister.com*. Mike's homepage can be found at *http://perlmeister.com*.

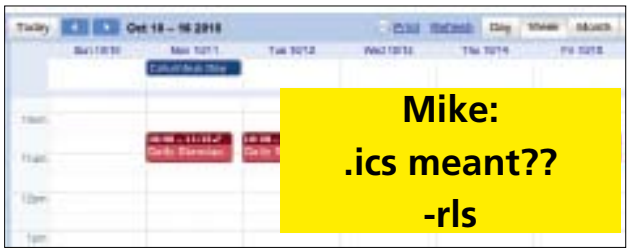

Figure 1: Google Calendar lists daily meetings, a weekly 1:1 meeting, and a public holiday on Monday.

```
RRULE:FREQ=WEEKLY;INTERVAL=2;BYDAY=WE
```

The calendar application uses this information to generate the meeting events as of a specific starting date (such as the current point in time) until a future time; it can also trigger actions such as notifications at these times.

Public Holidays in the Calendar

If Wednesday happens to be a public holiday, the fortnightly meeting with the boss will not take place, and I won't want to be alerted – if I can help it – to avoid disturbing my well-deserved peace of mind on this day of rest. Because public holidays follow complex rules, the Google server bundles them in another `.ics` file. Instead of meetings, the file includes full-day events if a day happens to be a public holiday.

Because I live and work in the United States, the "US Holidays" are the ones I need; if you happen to live in a different country, you need to go to *Other Calendars* | *Add* | *Browse Interesting Calendars*, select the holidays for your region, then press the *Subscribe* button to tell the calendar to import the holidays into your calendar.

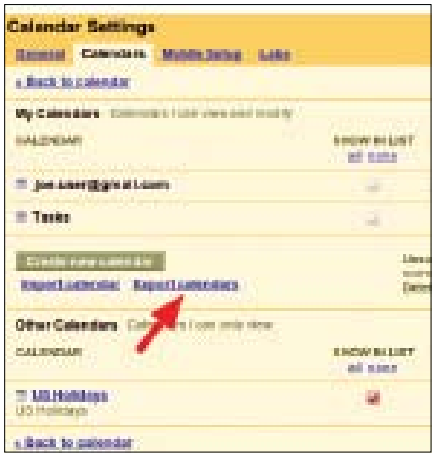

Figure 2: The Export function picks up the Google Calendar `.ics` file from the server.

After doing so, you will see a button labeled *ICAL* under *Other Calendars* | *Settings* | *XYZ Holidays* in the *Calendar Address* field, which takes you to the `.ics` file (Figure 5). Then you need to feed the holiday calendar (the `.ical` file in Figure 6) to your own calendar application, or the script, which will then take the holiday events into consideration and hide any meetings planned on these days.

Do-It-Yourself Calendar

When launched, the `ical-daemon` script parses all the `.ics` files in the `~/.ics-daemon/ics` directory and then uses the CPAN iCal::Parser module to create a data structure from them. The structure calculates the calendar events for today and organizes them in chronological order in the `@TODAYS_EVENTS` array.

Figure 3: The `.ics` file is exported as a `.zip` archive.

Line 10 in Listing 1 adds the CPAN local::lib module, which I have used frequently in the past, to support the installation of all additionally required CPAN modules below the user's home directory; this means that the user doesn't need to be root and doesn't need to disturb the way the package manager organizes things.

Lines 37 and 39 set the location for the logfile and the file in which to save the current process ID, `pid`. Line 52 initializes the Log4perl framework, which appends messages sent as `DEBUG`, `INFO`, or `WARN` information to the logfile. The App::Daemon module and the `daemonize()` function it exports ensure that the script understands the `ical-daemon start` and `ical-daemon stop` commands, which start and stop the daemon.

The script also uses a convenient approach to date calculations courtesy of the CPAN DateTime module. As an example of this, the module resets the time in the `$dt` DateTime object to the start of a day simply by calling `$dt->truncate( to => 'day' )`. DateTime also overloads comparative operators such as `<` and `>`, so that `$dt1 > $dt2` is precisely true if the time `$dt1` is later than the time `$dt2`.

Lines 20-22 define the 15-minute warning period before a meeting as a `DateTime::Duration` object and stores it in the `$ALERT_BEFORE` variable. Line 99 then subtracts this period from the meeting time and checks to see whether the current time has already advanced beyond this point.

Cinderella Effect

In the `while` loop beginning in line 70, the daemon regularly checks to see whether or not a meeting is due to begin in the next 15 minutes and, if so, calls the `ical-notify` script (Listing 2), which I will look at later. After this, it deletes the event from the array with the events of the day.

At midnight, the current date changes, and line 80 compares this with the day stored in `$CURRENT_DAY`. If it has changed, line 88 calls the `update()` function defined further down to parse all the `.ics` files and construct a new daily event array.

If line 99 notices that a meeting is closer than the grace period of 15 minutes, the `tap()` function exported by the CPAN Sysadm::Install module calls the `ical-notify` script in line 104. The FindBin module included in line 17 is one of the Perl distribution's standard functions; if needed, it can export a `$Bin`

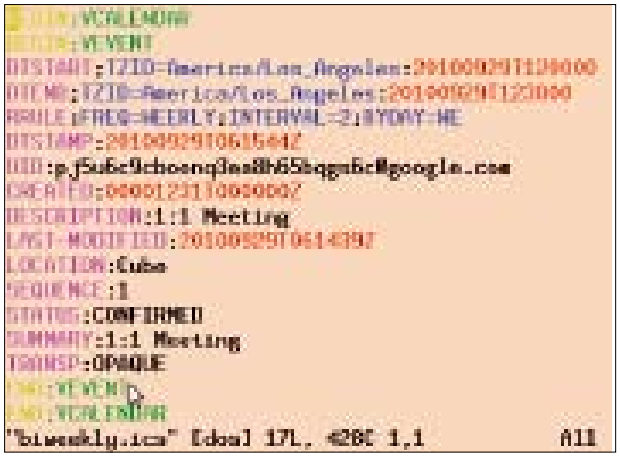

Figure 4: The `.ics` file exported from Google Calendar describes a meeting that takes place every other week.

Mike: Not {CN} eq "XYZ Holidays" ?? -rls

variable that specifies the directory in which the currently active script resides. The `tap()` in line 104 now references `$Bin` to find `ical-notify` in the same directory as the active daemon.

If the current day is a public holiday, the `update()` function notices this fact by calling `event_is_holiday()` (defined in lines 200-213); then, `update()` deletes all the appointments for the day and passes an empty array to the main program. To discover whether an event originated with the holiday calendar, `event_is_holiday()` checks to see whether the `ATTENDEE` field in the `CN` entry contains the "`US Holidays`" string; the matching lines in the `.ics` file with the holidays look like this:

```
ATTENDEE;...;CN=US Holidays;...
```

For another country's holiday calendar, this would be "`CN=XYZ Holidays`", where XYZ is the name of the country.

Forget Tomorrow!

The call to the CPAN iCal::Parser module constructor in line 129 expects two Date-

LISTING 1: ical-daemon (part1)

```
001 #!/usr/local/bin/perl -w
002 ############################
003 # ical-daemon - Parse .ics
004 #   files and send alerts on
005 #   upcoming events.
006 # Mike Schilli, 2010
007 # (m@perlmeister.com)
008 ############################
009 use strict;
010 use local::lib;
011 use iCal::Parser;
012 use Log::Log4perl qw(:easy);
013 use App::Daemon
014   qw(daemonize);
015 use Sysadm::Install
016   qw(mkd slurp tap);
017 use FindBin qw($Bin);
018
019 our $UPDATE_REQUESTED = 0;
020 our $ALERT_BEFORE =
021   DateTime::Duration->new(
022  minutes => 15);
023 our $CURRENT_DAY =
024   DateTime->today();
025 our @TODAYS_EVENTS = ();
026
027 my ($home) = glob "~";
028 my $admdir =
029   "$home/.ical-daemon";
030 my $icsdir = "$admdir/ics";
031
032 mkd $admdir
033   unless -d $admdir;
034 mkd $icsdir
035   unless -d $icsdir;
036
037 $App::Daemon::logfile =
038   "$admdir/log";
039 $App::Daemon::pidfile =
040   "$admdir/pid";
041
042 if (exists $ARGV[0]
043  and $ARGV[0] eq '-q')
044 {
045  my $pid =
046   App::Daemon::pid_file_read(
047   );
048  kill 10, $pid;   # Send USR1
049  exit 0;
050 }
051
052 Log::Log4perl->easy_init(
053  {
054   level => $DEBUG,
055   file =>
056     $App::Daemon::logfile
057  }
058 );
059
060 $SIG{USR1} = sub {
061  DEBUG "Received USR1";
062  $UPDATE_REQUESTED = 1;
063 };
064
065 $UPDATE_REQUESTED =
066   1;    # bootstrap
067
068 daemonize();
069
070 while (1) {
071  my $now =
072    DateTime->now(
073   time_zone => 'local');
074
075  my $today =
076    $now->clone->truncate(
077   to => 'day');
078
079  if ($UPDATE_REQUESTED
080   or $CURRENT_DAY ne $today)
081  {
082
083   $UPDATE_REQUESTED = 0;
084   $CURRENT_DAY      = $today;
085
086   DEBUG "Updating ...";
087   @TODAYS_EVENTS =
088     update($now);
089   DEBUG "Update done.";
090  }
091
092  if (scalar @TODAYS_EVENTS) {
093   my $entry =
094     $TODAYS_EVENTS[0];
095
096   DEBUG
097 "Next event at: $entry->[0]";
098
099   if ($now > $entry->[0] -
100    $ALERT_BEFORE)
101   {
102    INFO "Notification: ",
103    "$entry->[1] $entry->[0]";
104    tap "$Bin/ical-notify",
105      $entry->[1],
106      $entry->[0];
107    shift @TODAYS_EVENTS;
108    next;
109   }
110  }
111
112  DEBUG "Sleeping";
113  sleep 60;
114 }
115
116 ############################
117 sub update {
118 ############################
119  my ($now) = @_;
120
121  my $start =
122    $now->clone->truncate(
123   to => 'day');
124  my $tomorrow =
125    $now->clone->add(
126   days => 1);
127
128  my $parser =
129    iCal::Parser->new(
```

Time objects that define the time window for the current day as the period from midnight to midnight. This helps iCal::Parser generate the current day's events from recurring meetings and thus removes the need to extrapolate these events until the end of time. Once the clock strikes 12 midnight, the daemon refreshes its data anyway for the period of one day only.

The `parse()` method in line 141 devours any `.ics` files it finds, including the collection of holidays, and adds the newly discovered meeting data to the existing iCal::Parser object. The last call returns a reference to a hash, which lists events for, say, 10/11/2010 (mm/dd/yyyy) in a hash entry that looks like this:

```
$hash->{2010}->{10}->{11}.
```

If line 162 reveals that one event is a holiday, line 167 outputs a warning, and `update()` passes an empty event array back to the main program; a public holiday has priority over any other entries. If it's a business day instead, `update()` extracts the values for `DTSTART` and `DESCRIPTION` from the iCal data and appends the starting time for the meeting (in the form of a DateTime object) and the topic of the meeting to the `@events` array. Line 192 sorts the meetings in ascending order by start time, and line 196 passes the plan for the day to the main program in the form of an array.

Figure 5: The `.ics` file with the holiday events can be retrieved by clicking on the Google server's *ICAL* button.

Taking Minutes

To allow users to check what the daemon is doing, it calls on Log4perl to log all of its activities in the `~/.ical-daemon/log` file (Figure 7).

If you feel like optimizing the algorithm, the daemon could go to sleep until 15 minutes before the next meeting (or until the start of the next day), instead of just sleeping for one minute. However, log messages once a minute won't cost too much, and they do give you more precise details of when the

LISTING 1: settingfile.ini (part2)

```
130   start => $start,
131   end   => $tomorrow
132    );
133
134  my $hash;
135
136  for
137    my $file (<$icsdir/*.ics>)
138  {
139   DEBUG "Parsing $file";
140   $hash =
141     $parser->parse($file);
142  }
143
144  my $year  = $now->year;
145  my $month = $now->month;
146  my $day   = $now->day;
147
148  if (!exists $hash->{events}
149   ->{$year}->{$month}
150   ->{$day})
151  {
152   return ();
153  }
154
155  my $events =
156    $hash->{events}->{$year}
157    ->{$month}->{$day};
158
159  for my $key (keys %$events)
160  {
161   if (
162    event_is_holiday(
163     $events->{$key}
164    )
165     )
166   {
167    WARN
168  "No alerts today (holiday)";
169    return ();
170   }
171  }
172
173  my @events = ();
174
175  for my $key (keys %$events)
176  {
177   next
178     if $now > $events->{$key}
179      ->{DTSTART};
180
181   # already over?
182
183   push @events,
184     [
185    $events->{$key}
186      ->{DTSTART},
187    $events->{$key}
188      ->{DESCRIPTION},
189     ];
190  }
191
192  @events = sort {
193   $a->[0] <=> $b->[0]
194  } @events;
195
196  return @events;
197 }
198
199 ############################
200 sub event_is_holiday {
201 ############################
202  my ($event) = @_;
203
204  return undef unless
205   exists $event->{ATTENDEE};
206
207  if ($event->{ATTENDEE}->[0]
208   ->{CN} eq "US Holidays")
209  {
210   return 1;
211  }
212  return 0;
213 }
```

Figure 6: All US public holidays in a `.ics` file.

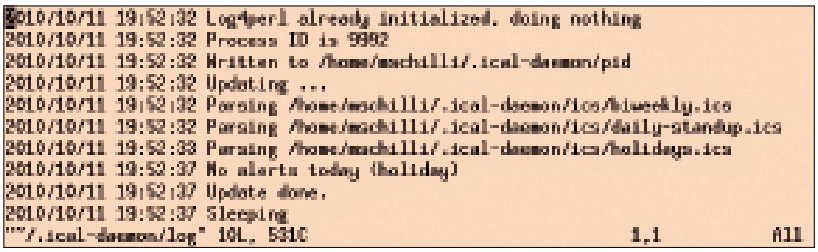

Figure 7: The daemon writes to the logfile to take minutes of current events.

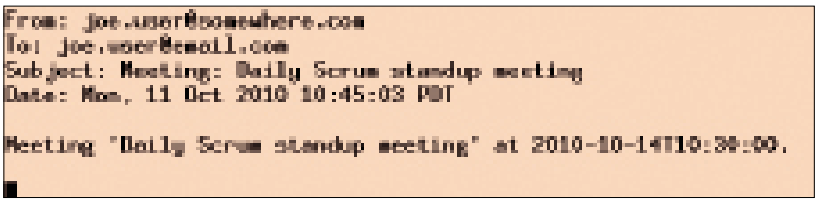

Figure 8: An email alerts the user to the fact that a meeting starts in 15 minutes.

daemon was active, when it was stopped, or – heaven forbid – when it crashed.

Poking the Daemon

To remove the need for the daemon to check the timestamps of the `.ics` files continually to see if new files have arrived – or old files have disappeared – the user can wake up the daemon manually by sending it a Unix signal. When the daemon receives the `USR1` signal, the signal handler in lines 60-63 sets the global `$UPDATE_REQUESTED` variable.

The next time the infinite `while` loop in lines 70-114 executes, the daemon notices the changed value and refreshes its internal data structures with the current `.ics` files. To remove the need for the user to know the PID of the daemon process in order to issue a `kill -USR1 pid` command from the command line, a call to the daemon with `ical-daemon -q` handles this. After sending the signal, the script immediately quits because of the `exit` command in line 49 without starting another daemon.

Because the daemon is implemented using the CPAN App::Daemon module, it stores its PID in a file, which is easy to locate by calling `App::Daemon::pid_file_read()` in line 46.

Email Wake-Up Call

Google offers all kinds of notification services, from pop-ups to text messages, but `ical-daemon` can run any script you want. My original idea was to use instant messages via Yahoo's new Messenger Web API [3], but there simply wasn't enough space here to cover this.

Maybe I'll come back to the subject in a future issue, once I've fought my way through the OAuth jungle.

Instead, the `ical-notify` script uses the CPAN Mail::DWIM module, which sends a message via the local SMTP daemon on port 25. Attentive readers might recall the dynamic tunnel mailer I built [4], but a normal Sendmail or Postfix process will do the trick as well. Figure 8 shows the mail that reaches the user 15 minutes before the meeting is due to start.

For the installation, you need to download the CPAN modules referred to here and preferably use local::lib to install them. If you need another country's holidays instead of the US holidays, you will also need to replace the "`US Holidays`" text string in line 154 of Listing 1 with the name of your region's holiday calendar. And despite rumors to the contrary, even if you're working in Europe, this won't actually give you many more days off work than I enjoy here in the US. ■■■

LISTING 2: ical-notify

```
01 #!/usr/local/bin/perl -w
02 ############################
03 # ical-notify - Email
04 #       calendar notification
05 # Mike Schilli, 2010
06 # (m@perlmeister.com)
07 ############################
08 use strict;
09 use local::lib;
10 use Mail::DWIM qw(mail);
11
12 my ($agenda, $time) = @ARGV;
13
14 die "usage: $0 time agenda"
15   unless defined $time;
16
17 mail(
18   to => 'm@perlmeister.com',
19   subject =>
20     "Meeting: $agenda",
21   text =>
22     "Meeting '$agenda' at " .
23     "$time.",
24   transport   => "smtp",
25   smtp_server => "localhost",
26 );
```

Mike: line 208 of Listing 1?? -rls

INFO

[1] Listings for this article: *http://www.linuxpromagazine.com/Resources/Article-Code*

[2] iCalendar: *http://en.wikipedia.org/wiki/ICalendar*

[3] Yahoo! Messenger IM API: *http://developer.yahoo.com/messenger/guide/ch02.html*

[4] "Drilling SSH Tunnels" by Mike Schilli, *Linux Magazine*, August 2010, pp. 48-54