Using D-Bus for automatic backups

Get on D-Bus

A Perl daemon starts an automatic backup with a progress indicator on the desktop when D-Bus reports that a USB stick has been plugged in. *By Michael Schilli*

I am loath to admit that my Ubuntu laptop has been gathering dust since the people at work handed me a MacBook about six months back. Can I just say two things in my own defense? A sleeping MacBook wakes up correctly in 99 of 100 cases and is ready for use within five seconds – including wireless. And, to create a backup, you just plug in a preconfigured USB backup disk, sit back, and watch the backup take place without moving a muscle (Figure 1).

You could say this is just a gimmick, but features like these liven up the daily grind, and if you don't have them, you can get all kinds of nasty withdrawal symptoms. Fortunately, state-of-the-art Linux desktops can trigger actions like these, too.

D-Bus, as used by Gnome, and now also by KDE, provides a practical communications channel between various applications, without the applications needing to know about each other. For example, when the Hardware Abstraction Layer (HAL) notices that the user has plugged in a USB stick, it sends a message on the D-Bus. Other applications, like the Gnome Desktop, pick up the message from the bus and then mount the stick in the `/media` directory (e.g., with Ubuntu) and open a "File Browser" window on the desktop.

Riding the System Bus

Even a scripting language, such as Perl, can ride the D-Bus thanks to the CPAN Net::DBus module. In Listing 1, line 12 selects the system bus, which collects and propagates system global messages independently of the active desktop session. Alternatively, D-Bus also offers the session bus that transports the data for the current user session. The `get_service()` method queries the bus object for the `"org.freedesktop.Hal"` service, which stores the HAL data in the `freedesktop.org` hierarchy, the D-Bus mother ship. The back-to-front notation helps organize the hierarchy and is well-known from the world of Java.

The `get_object()` in line 17 then uses this service to try to retrieve a HAL manager class object. In high-level languages, D-Bus often provides its services in the form of objects whose methods send or receive the bus data. The HAL Manager has a `GetAllDevices()` method, which returns a descriptive string for each connected device that HAL can identify. The `for` loop at the end prints them all out, as shown in Figure 2.

Listener on the Wall

While Listing 1 contacts the HAL Manager's Remote object to enumerate the devices registered thus far, you need a client who subscribes and keeps listening to the news on the bus to automagically launch a backup when a device is plugged in.

Because the documentation for some parts of sent D-Bus messages can be woefully lacking, it is often simpler to use a tool like `dbus-monitor`, which is included with the dbus package. When launched at the command line, this tool commandeers all D-Bus messages and outputs them when they arrive. Among other things, Figure 3 shows that a `MountAdded` event is passed to `dbus-monitor` when a user plugs in a USB stick. According to `dbus-monitor`, the service responsible for this message is `org.gtk.Private.GduVolumeMonitor`, which uses the `org.gtk.Private.RemoteVolumeMonitor` interface to serve up the `/org/gtk/Private/RemoteVolumeMonitor` object.

This event on the session bus no doubt originates with an application from the Gnome world that mounts the USB stick in `/media` and informs the listeners on the D-Bus of the event.

Backup Commando

To capture these events without drowning in a flood of irrelevant bus gossip, the backup daemon script, `dbus-mount-watcher`, registers with the D-Bus in Listing 2

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*.

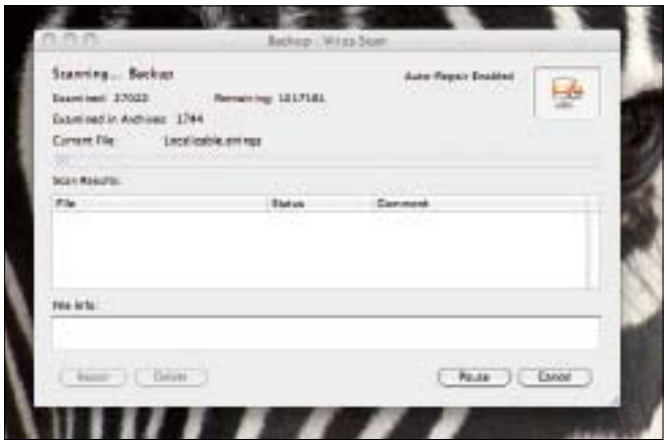

Figure 1: The MacBook immediately launches the "Time Machine" backup utility when the user plugs in the USB backup drive.

```
$ cg/hal-status
/org/freedesktop/Hal/devices/volume_uuid_A840_E2B3
/org/freedesktop/Hal/devices/storage_serial_Kingston_DT_101_G2_0013729820
AE8175760062_0_0
/org/freedesktop/Hal/devices/usb_device_930_6545_0013728820
if0_scsi_host_0_scsi_device_lun0_scsi_generic
/org/freedesktop/Hal/devices/computer
/org/freedesktop/Hal/devices/volume_uuid_8bbcdc5e_fb21_450e_
5fc
/org/freedesktop/Hal/devices/volume_part4_size_1024
/org/freedesktop/Hal/devices/volume_uuid_5aaa16d5_5274_4b2b_
55d
```

Figure 2: Net::DBus connects with the system D-Bus and outputs all hardware components detected by the HAL Manager.

and then subscribes to only *MountAdded* messages.

To see whether the user plugged in the known backup stick and not some other USB device, the script looks for the preset UUID A840-E2B3, which is the stick identification previously revealed by the D-Bus monitor. As soon as the daemon script recognizes the stick, it launches the `gtk2-backup` backup application in Listing 3 with a Gtk front end to initiate the backup procedure and display its current status on the desktop with a progress indicator.

LISTING 1: hal-status

```
01 #!/usr/local/bin/perl -w
02 ############################
03 # hal-status -- Get HAL
04 #   status via D-Bus
05 # Mike Schilli, 2011
06 # (m@perlmeister.com)
07 ############################
08 use strict;
09 use Net::DBus;
10
11 my $bus =
12   Net::DBus->system();
13 my $hal = $bus->get_service(
14  "org.freedesktop.Hal");
15
16 my $manager =
17   $hal->get_object(
18 "/org/freedesktop/Hal/Manager",
19 "org.freedesktop.Hal.Manager"
20   );
21
22 my $devices =
23   $manager->GetAllDevices();
24
25 for my $device (@$devices) {
26  print "$device\n";
27 }
```

Screenless Daemon

Because the daemon script runs in the background, thanks to the CPAN App::Daemon module, it is unaware of terminals or X server displays. To tell the graphical backup where to draw its GUI, the command in line 54 therefore defines the `DISPLAY` variable values as `:0.0` – that is, the first display on the active computer's X server.

The daemon script is launched by typing `dbus-mount-watcher start`, after which it disappears into the background thanks to the `daemonize()` method exported by App::Daemon. The user sees the command-line prompt return shortly after the launch. The daemon logs its activity in `/tmp/dbus-mount-watcher.log` (Figure 4).

The `dbus-mountwatcher stop` command terminates the daemon. For debugging purposes, the `-X` switch gives you the option of launching the daemon in the foreground (but log data are always written to the logfile), and `status` lets you query the daemon's status if you're in doubt about whether it's up or down.

The `connect_to_signal()` method in line 37 of Listing 2 assigns the `'MountAdded'` event on the session bus to the `mount_added()` callback in line 43. When an event is captured, the Net::DBus framework makes sure all the parameters are listed in the `dbus-monitor` output in Figure 3 are passed to the callback. The third parameter is thus a reference to an array, and the fifth element in the array is the USB stick mount point below the `/media` directory (Figure 5).

The `system()` call in line 58 of Listing 2 calls the graphical backup script `gtk2-backup` in Listing 3 and hands over the mount point as `file:///media/XXX`. The script's GUI pops up and immediately draws a red progress indicator on the screen after launching the backup process (Figure 6).

To prevent the daemon from stopping after registering with the D-Bus, and to instead keep it running indefinitely while time handling D-Bus events, line 63 defines a "Reactor." This object has a `run()` method that binds the daemon to the D-Bus for all time.

Building Boxes

Listing 3 accepts the mount point for the identified USB stick, removes its `file://` precursor in line 25, and then uses the command in line 34 to define the simple backup method: the `tar` command collects all the files below the `$src_dir` directory (line 15) and writes the resulting tarball to the USB stick. To prevent overwriting, the script creates a file name for the current data in a *YYYYMMDD*`.tgz` format. If you plug in the stick more than once a day, you need to modify this to add hours and minutes.

```
$ dbus-monitor
...
signal sender=:1.22 -> dest=(null destination) serial
=15 path=/org/gtk/Private/RemoteVolumeMonitor; interf
ace=org.gtk.Private.RemoteVolumeMonitor; member=Mount
Added
   string "org.gtk.Private.GduVolumeMonitor"
   string "0x90ff4d8"
   struct {
      string "0x90ff4d8"
      string "7.7 GB Filesystem"
      string ", GThemedIcon drive-removable-media-usb
drive-removable-media drive-removable drive"
      string ""
      string "file:///media/A840-E2B3"
      boolean true
      string "0x9100df0"
      array [
      ]
   }
$
```

Figure 3: News of a recently plugged in and mounted USB stick is propagated on the D-Bus.

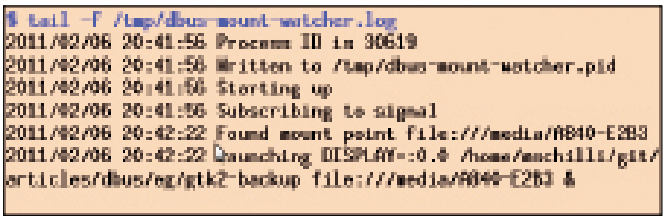

Figure 4: The logfile shows the USB stick was identified 20 seconds after launching the script and the backup process was initiated.

The GUI layout comprises an upper section with the progress indicator and a lower section with a button that reads *Cancel* during the backup. This button toggles to a success message after the backup completes. Because widgets, like progress indicators, can't be displayed directly in a Gtk2::Window class window, you need a Gtk2::VBox to contain them. The `pack_start()` method inserts the progress bar and the button into the container.

Figure 5: Net::Dbus calls the callback for the MountAdd signal with these parameters.

If the backup is too slow for your liking, you can interrupt the process by pressing the *Cancel* button. In this case, the script sends a Sigterm signal (numerical signal value 2) to the tar process; the process quits, which triggers an error in `close()` in line 105 and thus dismantles the GUI before shutting down the backup script.

Progress with a Trick

To make sure the progress indicator gives a fair representation of the backup status, the CPAN File::Finder module first collects all the files (type "`f`") below the `$src_dir` directory and all the subdirectories in line 31, and counts them using the `scalar` operator on the resulting array.

With tar running in verbose mode, the `while` loop in lines 88-103 picks up each new line of output and thus has a very good idea how many files tar has backed up. Line 96 reports the ratio of finished files to the total number of files to the progress indicator, with line 92 adding the *Backup Progress (XX/YY)* text. The Gtk2 construct with the `main_iteration` method in line 99 refreshes the front end each time the progress indicator moves; otherwise, buffering would prevent any progress from being displayed. When the `tar` command completes, line 108 outputs a success message in the button below the bar; clicking the button (or pressing Enter) terminates the program.

To make sure the data written by tar ends up on the USB stick rather than being cached somewhere in the operating system layers, you should `umount` the stick, either at the command line or in the file manager, before you unplug it.

The GUI starts to run on entering the main event loop (`Gtk2->main`) in line 77, where it then waits for user input. Because you want the backup program to run automatically without waiting for

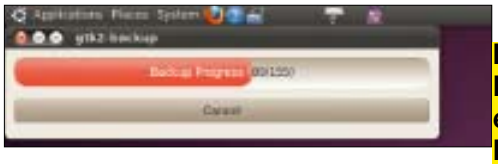

Figure 6: Automatic backup is launched in Ubuntu immediately after plugging in the USB stick.

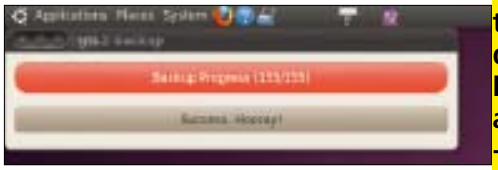

Figure 7: The backup was successful, and the tarball is now safely on the USB stick.

Layout: I shortened the Fig6 caption box and moved the fig down. Please adjust. -rls

LISTING 2: dbus-mount-watcher

```
01 #!/usr/local/bin/perl -w
02 ##############################
03 # mount-watcher
04 # Mike Schilli, 2011
05 # (m@perlmeister.com)
06 ##############################
07 use strict;
08 use Net::DBus;
09 use Net::DBus::Reactor;
10 use App::Daemon;
11 use FindBin qw($Bin);
12 use Log::Log4perl qw(:easy);
13
14 use App::Daemon
15   qw( daemonize );
16 daemonize();
17
18 INFO "Starting up";
19
20 my $BACKUP_STICK =
21   "file:///media/A840-E2B3";
22 my $BACKUP_PROCESS =
23   "$Bin/gtk2-backup";
24
25 my $notifications =
26   Net::DBus->session
27   ->get_service(
28 "org.gtk.Private.GduVolumeMonitor"
29   )->get_object(
30 "/org/gtk/Private/
   RemoteVolumeMonitor",
31 "org.gtk.Private.
   RemoteVolumeMonitor",
32   );
33
34 INFO "Subscribing to signal";
35
36 $notifications
37   ->connect_to_signal(
38  'MountAdded',
39  \&mount_added
40   );
41
42 ##############################
43 sub mount_added {
44 ##############################
45  my ($service, $addr, $data)
46    = @_;
47
48  INFO "Found mount point ",
49       "$data->[4] ";
50
51  if ($data->[4] eq
52   $BACKUP_STICK)
53  {
54   my $cmd = "DISPLAY=:0.0 "
55     . "$BACKUP_PROCESS "
56     . "$data->[4] &";
57   INFO "Launching $cmd";
58   system($cmd );
59  }
60 }
61
62 my $reactor =
63   Net::DBus::Reactor->main();
64 $reactor->run();
```

the user to click, line 73 sets a timer. The timer calls the `start()` function defined in lines 80-113 as a low priority task, `Glib::G_PRIORITY_LOW`. The Glib kernel will not start this task until it is sure there are no more GUI draw tasks to complete.

The timer callback `start()` expects a single parameter, which is a reference to the `$pbar` progress bar widget. It returns the value `Glib::SOURCE_REMOVE` after completing its work; otherwise, the timer would repeat the callback after another timeout, and the backup would restart.

Installation

All the main Linux distributions include the `dbus` package. The `gdbusviewer` tool, another diagnostics tool besides `dbus-monitor`, can be installed on Ubuntu with the `gq4-dev-tools` collection. The required Perl modules are available as libdatetime-perl, libfile-finder-perl, libgtk2-perl, libglib-perl, libapp-daemon-perl, liblog-log4perl-perl, and libnet-dbus-perl from the Ubuntu repositories.

The daemon is then launched by typing `dbus-mount-watcher start`; if you want to start the daemon automatically when you reboot, you can add it below `/etc/init.d/` and register with `update-rc.d`. The graphical backup script should reside in the same directory as the daemon, or you could use an absolute path to call it from the daemon.

D-Bus has much more to offer than the tricks discussed in this column. Applications like the Pidgin instant messenger client or the Rhythmbox media player are tightly integrated with D-Bus, so, besides being simply monitored, they can be remote controlled via this smart communication mechanism [4]. ■■■

INFO

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

[2] Introduction To D-Bus: *http://www.freedesktop.org/wiki/IntroductionToDBus*

[3] "D-Bus with Perl" by Emmanuel Rodriguez: *http://bratislava.pm.org/presentation/dbus/perl-dbus.pdf*

[4] Pidgin integration with D-Bus: *http://developer.pidgin.im/wiki/DbusHowto*

LISTING 3: gtk2-backup

```
001 #!/usr/local/bin/perl -w
002 ############################
003 # gtk2-backup
004 # Mike Schilli, 2011
005 # (m@perlmeister.com)
006 ############################
007 use strict;
008 use File::Finder;
009 use Glib qw/TRUE FALSE/;
010 use Gtk2 '-init';
011 use DateTime;
012
013 my $PID;
014 my $tar = "tar";
015 my $src_dir =
016   "/home/mschilli/test";
017 my $ymd =
018   DateTime->now->ymd('');
019
020 my ($stick_dir) = @ARGV;
021
022 if (!defined $stick_dir) {
023  die "usage: $0 stick_dir";
024 }
025 $stick_dir =~ s#^file://##;
026
027 my $dst_tarball =
028   "$stick_dir/$ymd.tgz";
029
030 my $NOF_FILES =
031   scalar File::Finder->type(
032  "f")->in($src_dir);
033
034 my $CMD = "$tar zcfv " .
035      "$dst_tarball $src_dir";
036
037 my $window =
038   Gtk2::Window->new(
039  'toplevel');
040
041 $window->set_border_width(
042  10);
043 $window->set_size_request(
044  500, 100);
045
046 my $vbox =
047   Gtk2::VBox->new(TRUE, 10);
048 $window->add($vbox);
049
050 my $pbar =
051   Gtk2::ProgressBar->new();
052 $pbar->set_fraction(0);
053 $pbar->set_text("Progress");
054 $vbox->pack_start($pbar,
055  TRUE, TRUE, 0);
056
057 my $cancel =
058   Gtk2::Button->new(
059  'Cancel');
060 $vbox->pack_end($cancel,
061  FALSE, FALSE, 0);
062 $cancel->signal_connect(
063  clicked => sub {
064   kill 2, $PID
065     if defined $PID;
066   Gtk2->main_quit;
067  }
068 );
069
070 $window->show_all();
071
072 my $timer =
073   Glib::Timeout->add(10,
074  \&start, $pbar,
075  Glib::G_PRIORITY_LOW);
076
077 Gtk2->main;
078
079 ############################
080 sub start {
081 ############################
082  my ($pbar) = @_;
083
084  $PID = open my $fh,
085    "$CMD |";
086
087  my $count = 1;
088  while (<$fh>) {
089   chomp;
090   next if m#/$#;  # skip dirs
091
092   $pbar->set_text(
093        "Backup Progress "
094      . "($count/$NOF_FILES)"
095   );
096   $pbar->set_fraction(
097    $count / $NOF_FILES);
098
099   Gtk2->main_iteration while
100    Gtk2->events_pending;
101
102   $count++;
103  }
104
105  close $fh
106    or die "$CMD failed ($!)";
107
108  $cancel->set_label(
109   "Success. Hooray!");
110  undef $PID;
111
112  return Glib::SOURCE_REMOVE;
113 }
```