Unboxing Power Notifications and Vector Icons in ChromeOS

Rajat Khandelwal

Intel Technology India Pvt. Ltd., Bengaluru, India

---***---

Abstract—ChromeOS houses ash tray which contain battery, volume and other icons. This ash tray keeps on updating based on the values received by the OS framework. The battery keeps on updating with many symbols like low power charger connected, no charger connected, no battery, battery charging, etc. Similarly, we have a notification mechanism in ChromeOS which displays us information about the events happening currently like a device getting hot plugged in/out or a battery being removed or a charger being plugged in, etc. We also get notifications that concern the time remaining estimate of the battery charging or battery emptying. The ACPI kernel framework of battery exposes sysfs attributes which represents multiple types of information. Similarly, if a mouse/keyboard (external device) gets plugged in, or a low power charger gets plugged in, or a AC charger gets plugged in, respective directories are exposed to sysfs representing each type of device. We also get information like is the charger dual power role, or the maximum/minimum power/energy consumption of the source. This entire information gets conveyed to UI through DBUS mechanism. There also exists a daemon in power domain which polls this information and transmits it to ChromeOS through DBUS signaling.

Keywords—DBUS, Notifications, Battery, Vector icons, User interface, Power

I. INTRODUCTION

All the operating systems in general have two mechanisms to notify the user about the battery and power capabilities. Those are notifications and tray icons.

In windows platform, we have a rectangular battery icon which is laid out horizontal. If battery is too low, a red cross sign appears in front of us and a notification displaying "Battery low" is presented in UI. Similarly, if the battery is being charged, a charger icon gets portrayed in the status bar.

ChromeOS also delivers these features in a very sophisticated way. There exists a daemon in power domain which polls the power related information every 30 seconds and notifies the information to ChromeOS via DBUS signaling. The mechanism of DBUS signaling is explained later in the paper.

We can find the logs pushed by the power manager class of power domain in sysfs. These are located in /var/log/power_manager/powerd.LATEST.

II. THE INITIALIZATION

powerd daemon starts from its Init() function which is defined in daemon.cc file. This function then creates a power supply class by calling initialization function of power supply class. The pictorial representation is given below.

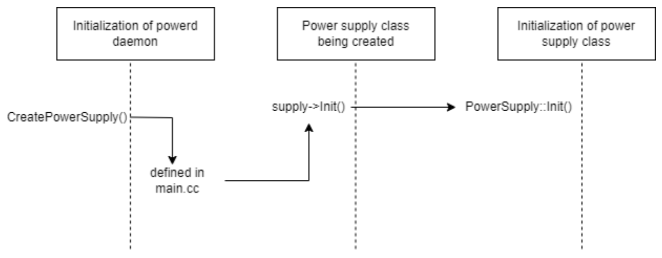
Initialization of powerd daemon
Power supply class being created
Initialization of power supply class
CreatePowerSupply()
defined in main.cc
supply->Init()
PowerSupply::Init()

Fig.1. Initialization of powerd daemon

Daemon initialization starts with CreatePowerSupply method which is actually defined in main.cc. This function then calls the initialization function of power supply class. This is how the power supply class finally gets initialized.

III. INITIALIZATION OF POWER SUPPLY CLASS

There are various variables which get initialized during the initialization of power supply class. These variables are listed below:

A. Power supply path: The sysfs directory exposed which provides battery, charger and connected devices' attributes gets initialized to /sys/class/power_supply.

B. Udev observer: The power supply class gets added to the udev subsystem as an observer.

C. Poll delay: This represents the amount of time to wait before updating the power status again after an update.

D. Initial poll delay: This represents the amount of time to wait before updating the power status again after an update if the number of samples is less than "max current samples".

E. Battery stabilized after startup delay: This represents the amount of time to wait after startup before assuming that the current can be used in battery time estimates and the charge is accurate.

F. Battery stabilized after line power connected delay: This represents the amount of time to wait after a power source gets connected before assuming that the current can be used in battery time estimates and the charge is accurate.

G. Battery stabilized after line power disconnected delay: This represents the amount of time to wait after a power source gets disconnected before assuming that the current can be used in battery time estimates and the charge is accurate.

H. Battery stabilized after resume delay: This represents the amount of time to wait after a resume event before assuming that the current can be used in battery time estimates and the charge is accurate.

I. Full factor: This represents the full factor of the battery and is used to calculate display battery percentage.

When power supply class gets initialized, it calls two important functions – defer battery sampling and schedule poll.

The complete pictorial representation of the power supply class initialization is given below:

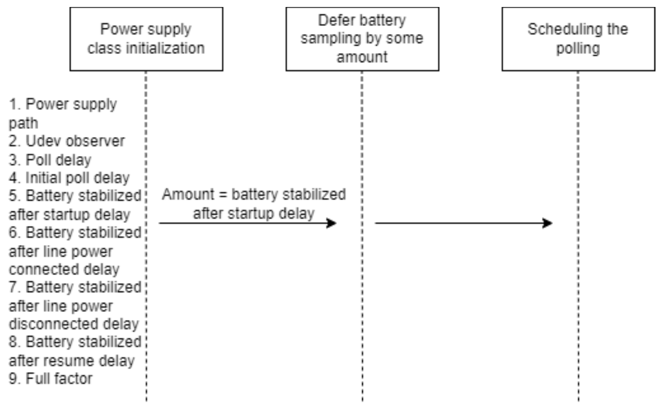
Power supply class initialization
Defer battery sampling by some amount
Scheduling the polling
1. Power supply path
2. Udev observer
3. Poll delay
4. Initial poll delay
5. Battery stabilized after startup delay
6. Battery stabilized after line power connected delay
7. Battery stabilized after line power disconnected delay
8. Battery stabilized after resume delay
9. Full factor
Amount = battery stabilized after startup delay

Fig. 2. Initialization of power supply class

IV. DEFERRING BATTERY SAMPLING

There exists a battery stabilized timestamp which indicates the exact time battery was stabilized last. It has to be updated continuously.

The current time is calculated first. Now, the delay given in the function (here, battery stabilized after startup delay) is added to the current time and the maximum of the current battery stabilized timestamp and the addition of the above two quantities provides the new battery stabilized timestamp.

This also gets generated as a log in the power manager logs.

V. SCHEDULING THE POLLING

First, the current time is calculated. Now, we know that battery current and charge are stabilized at battery stabilized timestamp. But to poll them, we have to wait another 50 milliseconds (slack milliseconds) after battery stabilized timestamp.

If battery stabilized timestamp is greater than the current time, the next polling will begin in adding slack milliseconds to the difference of current time and battery stabilized timestamp.

If battery stabilized timestamp is less than the current time, that implies, it has not been updated yet. So, if the current samples are less than max current samples, the next polling will begin in "initial poll delay" time.

If battery stabilized timestamp is less than the current time, and the current samples are more than the max current samples, the next polling will begin in "poll delay" time.

This information gets logged in the power manager logs depicting the time after which the polling will start.

Now, the poll timer gets started. It will call a function – "OnPollTimeout" when the time reaches polling time.

The pictorial representation of the complete polling mechanism is given below:

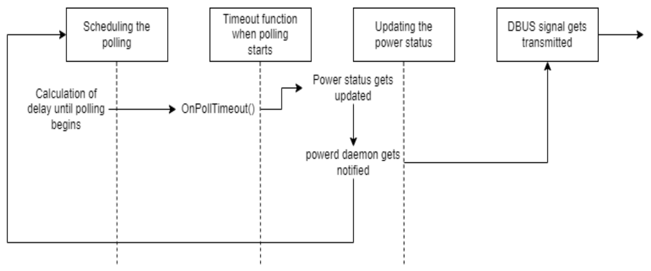
Scheduling the polling
Timeout function when polling starts
Updating the power status
DBUS signal gets transmitted
Calculation of delay until polling begins
OnPollTimeout()
Power status gets updated
powerd daemon gets notified

Fig. 3. Flow chart from polling up to DBUS transmission

We can gather from the figure that after updating the power status, powerd daemon gets notified that the power status has been updated. This happens in order for the daemon to log the power status in the power manager logs.

There is one more point to consider. Since, we had already discussed that powerd polls periodically, so after the daemon gets notified, the parameter 'delay' again gets updated and polling again gets scheduled.

Finally, the DBUS signal carrying a buffer with contents equal to the current power status gets transmitted to the ChromeOS to reflect the power status in notification and ash tray.

VI. UPDATING POWER STATUS

The power supply path was already initialized and was equal to /sys/class/power_supply. Now, the sysfs directory is iterated to poll battery, charger and devices connected.

If the "scope" parameter of sysfs directory reflects "Device", that implies, it is an external peripheral device and is not relevant to us.

If the "type" parameter of sysfs directory reflects "Battery", then the path gets appended in the list of battery paths.

If the "type" parameter of sysfs directory doesn't reflect "Battery", then its evident that it is a line power directory. So the line power attributes are now read.

After reading the line power attributes exposed in sysfs, the battery attributes are read via the battery path appended before.

Finally, the observed battery charge rate and battery time estimates get updated based on the attributes collected by reading from sysfs.

There is one more parameter which gets updated – "battery less than shutdown threshold".

We will discuss in the paper how each of these parameters are calculated and transmitted to UI through DBUS signaling.

VII. READING LINE POWER DIRECTORY

The following attributes of line power directory get read through the sysfs.

A. Line status: This attribute gets populates by reading the "status" field from the sysfs line power directory.

B. Supports dual role devices: Bidirectional/dual role ports export "status" field. Thus, this attribute gets populated by reading the line status. If line status exists, then it supports dual role else it doesn't.

C. Type of the connected device: This attribute gets populated by reading the "type" field from the sysfs line power directory.

D. Is the connected adapter dual role: If the type of the connected device is USB, then the "usb_type" field of the sysfs line power directory gets read. If this field ends with "PD_DRP", then the connected line power source is a dual power source.

E. Is the power supply online: This attribute indicates that if '0', nothing is connected, unless it is a dual role power device, in which case a value of '0' indicates that we are connected to a dual role power device but not sinking power. This attribute gets populated by reading the "online" field from the sysfs line power directory.

F. Manufacturer: This attribute represents the manufacturer of the line power source and is populated by reading the "manufacturer" field from the sysfs line power directory.

G. Model name: This attribute represents the model name of the line power source and is populated by reading the "model_name" field from the sysfs line power directory.

H. Voltage max.: This field represents the max voltage given by the power source and is populated by reading the "voltage_max_design" field from the sysfs line power directory and multiplying it with a double scale factor of 0.000001.

I. Current max.: This field represents the max current given by the power source and is populated by reading the "current_max" field from the sysfs line power directory and multiplying it with a double scale factor of 0.000001.

J. Power max.: This attribute represents the max power supplied by the power source and is populated by multiplying voltage max by current max.

K. Voltage now: This attribute represents the current voltage and is populated by reading "voltage_now" field from the sysfs line power directory and multiplying it with a double scale factor of 0.000001.

L. Current now: This attribute represents the current and is populated by reading "current_now" field from the sysfs line power directory and multiplying it with a double scale factor of 0.000001.

M. Is the charger low power USB charger: If the charger connected is not dual role power, and the type of charger connected is either USB or USB_ACA or USB_CDP or USB_DCP, then the power source connected is a low power charger.

N. Is the charger low power dual role charger: If the charger connected is a dual role power charger and the max power is less than USB min AC power, then the power source connected is a dual role low power charger.

O. Is the charger a high power source: If the above two conditions don't hold, then the power source connected is a high power source.

VIII. READING BATTERY DIRECTORY

The following attributes of battery directory get read through the sysfs.

A. Is battery present: This attribute represents if a battery is present and is populated by reading "present" field from the sysfs battery directory. This field should not be equal to 0.

B. Status of the battery: This attribute represents the current status of the battery (charging or not) and is populated by reading "status" field from the sysfs battery directory.

C. Manufacturer: This attribute represents manufacturer of the battery and is populated by reading "manufacturer" field from the sysfs battery directory.

D. Model name: This attribute represents model name of the battery and is populated by reading "model_name" field from the sysfs battery directory.

E. Technology: This attribute represents technology of the battery and is populated by reading "technology" field from the sysfs battery directory.

F. Voltage now: This attribute represents current voltage of the battery and is populated by reading "voltage_now" field from the sysfs battery directory and multiplying it with a double scale factor of 0.000001.

G. Serial number: This attribute represents serial no of the battery and is populated by reading "serial_number" field from the sysfs battery directory.

H. Voltage min design: This attribute represents voltage min of the battery and is populated by reading "voltage_min_design" field from the sysfs battery directory and multiplying it with a double scale factor of 0.000001.

I. Voltage max design: This attribute represents voltage max of the battery and is populated by reading "voltage_max_design" field from the sysfs battery directory and multiplying it with a double scale factor of 0.000001.

ACPI has two different battery types: charge battery and energy battery. Charge battery exposes current now in A, charge now, charge full and charge full design in Ah. Energy battery exposes power now in W and energy now, energy full and energy full design in Wh.

There are four parameters that need to be read: charge, charge full, charge full design and energy in order to make calculations.

If the battery is charge battery, charge, charge full and charge full design are calculated by reading the sysfs power directory attributes and energy is calculated by multiplying the charge now with nominal voltage.

If the battery is energy battery, energy, energy full and energy full design are calculated by reading the sysfs power attributes. With these, charge, charge full and charge full design can also be calculated by dividing these parameters with nominal voltage.

If the sysfs battery directory exposes "power_now" field, then the current is calculated by reading the power now and dividing it by voltage.

If the sysfs battery directory exposes "current now" field, then the current is calculated by reading the field.

IX. UPDATING BATTERY PERCENTAGE AND STATE

Battery percentage is calculated by dividing battery charge by battery charge full and multiplying the result by 100. This battery percentage is converted into display battery percentage before providing it to DBUS.

If the line power is on, and the battery charge is more than or equal to battery charge full multiplied by full factor, the status of the battery is termed as FULL.

If the line power is on, current is more than 0, and the battery status reflects charging, then the status of the battery is termed as CHARGING.

If none of the above conditions hold, status of the battery is termed as DISCHARGING.

X. BATTERY CHARGE RATE, ESTIMATE AND SHUTDOWN

If the status of the battery is CHARGING, the charge left to full is calculated by multiplying charge full by full factor and subtracting battery charge from it. Now, time to full can be calculated by rounding the product of 3600 and charge to full and dividing it by signed current.

Signed current is calculated by taking average of the current samples on line power (if on line power) or average of the current samples on battery (if on battery).

If the status of the battery is DISCHARGING, the time to empty is calculated by multiplying battery charge and nominal voltage and dividing it by the product of signed current and battery voltage and multiplying the result by 3600 and rounding it.

Similarly, battery time to shutdown is also calculated.

The parameter – is battery below shutdown threshold is also calculated from the above known parameters.

The entire power status collected so far is transmitted to UI through DBUS signal – PowerSupplyPollSignal.

XI. DBUS

DBUS is a system for IPC communication. On the high level, DBUS has a libdbus library which allows two applications to connect to each other to exchange messages or data. Built on libdbus is the message bus daemon to which several applications can connect to and is responsible for routing data from a sender to a receiver. There are two types of daemon instances – system wide and per session.

Objects/Object paths:

A. Just like any other programming framework DBUS abstracts the concept of object in a similar way with a base class.

B. Libdbus however provides an object path and not a native object.

C. Object path is similar to a sysfs path.

D. Why are object paths needed? High level bindings can name the instances defined of native objects and thus remote applications can refer to them in an abstract way. Eg. org/chromium/power_manager.

Methods and signals:

A. Members of the DBUS object.

B. Methods: which can be invoked on an object with input/output, arguments, etc.

C. Signals - a means of broadcasting a payload from an object to a respective observer ; in our case: PowerSupplyPoll signal is sent containing PowerSupplyProperties proto buffer as the payload.

Interfaces:

A. Each DBUS object supports one/more interfaces: basically a named group of methods and signals.

B. Represented by single name spaced string: org.freeDesktop.Notifications / org.chromium.PowerManager.

Bus names:

A. DBUS daemon assigns each application a name whenever it connects to it: unique connection name.

B. Starts with a ':' character: treat them like IP addresses.

C. Eg. suppose an object path com/company/notificationservice with an interface org.freeDesktop.Notifications could be given a name of :43-574.

D. So now applications send messages to bus names, object and interface to execute method calls.

Addresses:

A. In our DBUS case, server is the DBUS daemon as it listens to all applications which connect to it and thus are clients.

B. Address: where a server would listen.

C. For system wide messages, libdbus knows a well defined UNIX socket path: it will be most likely at /run/dbus/system_bus_socket.

D. For session messages, it reads an environment variable. [1]

It all comes down to: **Address → Name → Bus name → Object path → Interface → Method.**

A pictorial representation of DBUS is given below:

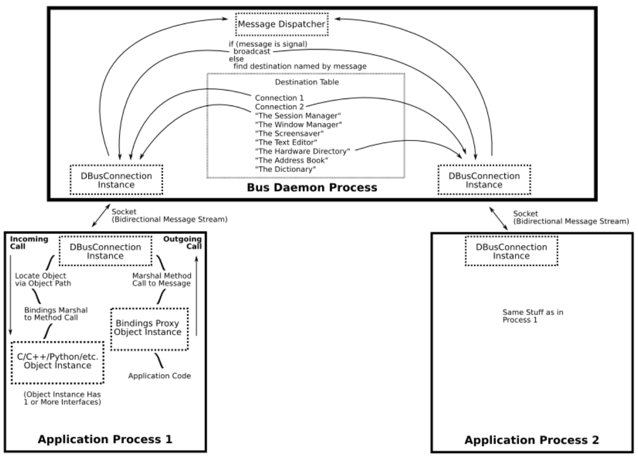

Fig. 4. DBUS Interface

XII. DBUS SIGNAL RECEIVE INTERFACE

The first function which gets called upon receiving the DBUS signal – PowerPollSupplySignal is PowerSupplyPollReceived() which is present in the power manager client of ash system class.

Then, the observer "power status class" gets called which in turn calls all its observers and their OnPowerStatusChanged() function.

There are two observers in the power domain of the power status class: power notification controller and tray power. These represent the notification handling and the icon handling in the ash tray.

The pictorial representation is given below:

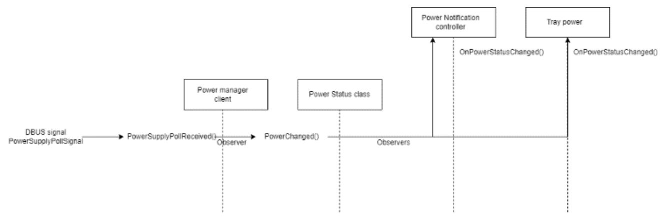

Fig. 5. DBUS signal receive interface

XIII. TRAY ICON

Icons in chrome are represented in the form of vector icons and are drawn with the help of Skia gfx library.

To describe a vector icon, we have to make a .icon file and put it in one of the appropriate vector_icons directory. There are a total of 3 such directories: component/vector_icons, ui/views/vector_icons, ash/resources/vector_icons.

After adding it in the .icon file, we need to add the filename in the respective BUILD.gn which in turn creates a constant to reference the icon.

For eg., unified_menu_battery_bolt.icon will have the constant name as kUnifiedMenuBatteryBoltIcon.

The location of all the icons concerning battery is ash/resources/vector_icons. Their names start with unified_menu_battery*.

To create a vector icon, we simply need to call the library function: CreateVectorIcon.

We have already seen that the function to be called in tray power is OnPowerStatusChanged(). This function is responsible for calculating the battery image information including the badge which needs to be put on the battery icon if necessary depending on the current parameters.

The power status carrying all the information about the type of charger connected and battery status is already transmitted via DBUS interface. This information is used to identify which badge needs to be put on the battery icon. The image of the badge can be found in the location of the icons concerning battery.

XIV. POWER NOTIFICATION CONTROLLER

The power status concerning the type of charger connected and the battery transmitted is transmitted via DBUS interface to the power status class. This information is used to determine if a notification needs to be presented.

For eg., if a low power USB charger is connected, the notification stating the same should be presented. Similar are cases for battery full, battery about to empty, etc.

Notifications in ChromeOS are created using CreateSystemNotification function.

ACKNOWLEDGMENT

I would like to acknowledge Intel for providing me this opportunity to conduct a research on power domain and UI interface.

REFERENCES

[1] David Wheeler, John Palmieri, Colin Walters, "D-Bus Tutorial", version 0.5.0

[2] Torbjorn Semb Dahl, Faruque Sarker, "Flexible Communication in Multi-robotic Control System using HEAD: Hybrid Event-Driven Architecture on D-Bus", September, 2010

[3] Google, "Chrome OS Power Management"

[4] Karunesh Johri, "D-Bus Tutorial", December 20, 2016