Introduction:

The hardware abstract layer is the implementation between the computer hardware and the software, HAL will let us do a lot of hardware specific tasks with simple C calls.

HAL is a system daemon that gets launched at startup level after DBus, because it depends on it, the library itself is easy to learn just by reading the header files, they are usually under /usr/include/hal, and they are two files, one is libhal.h which contains API for general hardware devices properties and the other is libhal-storage.h contains the API for hardware storage specific tasks.

There are three main concepts that should be explained before start coding:

LibHalContext:

LibHalContext is the HAL context that an application should use to talk to HAL.

UDI:

Unique device identifier, HAL uses associates with each piece of hardware a udi.

Key:

Key is a property that holds a value on a UDI, example a battery device can have the key "battery_is_charging" which will be true if the battery is charging false otherwise.

We will start with a simple example that asks HAL for some information about the system that we are on, to do this, the UDI used here is /org/freedesktop/Hal/devices/computer, see HAL spec for list of UDI, keys, etc...

```
#include <stdio.h>

#include <hal/libhal.h>
#include <dbus/dbus.h>

int main()
{
    DBusConnection *connection;
    DBusError error;

    LibHalContext *ctx;
    const char *udi = "/org/freedesktop/Hal/devices/computer";

    dbus_error_init(&error);
    connection = dbus_bus_get(DBUS_BUS_SYSTEM,&error);

    if ( dbus_error_is_set(&error) )
    {
        printf("Unable to connect to DBus: %s\n",error.message);
        dbus_error_free(&error);
        return 1;
    }

    ctx = libhal_ctx_new();
```

```
    if ( !libhal_ctx_set_dbus_connection(ctx, connection) )
    {
        printf("Error %s\n",error.message);
        dbus_error_free(&error);
        return 1;
    }

    if ( !libhal_ctx_init(ctx, &error) )
    {
        printf("Hal context initializing failure %s\n",error.message);
        return 1;
    }

    char *kernel_version =
    libhal_device_get_property_string(ctx, udi, "system.kernel.version", &error);

    if ( dbus_error_is_set(&error) )
    {
        printf("Error getting string property %s\n", error.message);
        dbus_error_free(&error);
        return 1;
    }

    char *power_management_type =
    libhal_device_get_property_string(ctx, udi, "power_management.type", &error);

    if ( dbus_error_is_set(&error) )
    {
        printf("Error getting string property %s\n", error.message);
        dbus_error_free(&error);
        return 1;
    }

    dbus_bool_t can_hibernate =
    libhal_device_get_property_bool(ctx, udi, "power_management.can_hibernate",
&error);

    if ( dbus_error_is_set(&error) )
    {
        printf("Error getting bool property %s\n", error.message);
        dbus_error_free(&error);
        return 1;
    }

    printf("Your system is running a kernel version = %s with power management
type %s\n",
           kernel_version, power_management_type);
    if ( can_hibernate )
    {
        printf("Your system can hibernate\n");
    }
    else
    {
        printf("Your system can't hibernate\n");
    }

    libhal_free_string(kernel_version);
```

```
    libhal_free_string(power_management_type);

    return 0;
}
```

To compile the above example run the following command:

```
gcc `pkg-config --libs --cflags hal dbus-1` hal-example.c -o hal-example
```

Finding out a device UDI:

Devices have capabilities in addition to their keys, depending on the capability and the keys attached with a UDI , we can figure out what is the type of such device, in the following example we will find out all the input devices.

Same beginning of the example above, so no need to repeat.

```
    int num_devices = 0;
    char **udis =
    libhal_find_device_by_capability(ctx, "input", &num_devices, &error);

    if ( dbus_error_is_set(&error) )
    {
        printf("Error getting bool property %s\n", error.message);
        dbus_error_free(&error);
        return 1;
    }

    if ( num_devices == 0 )
    {
        printf("No device found with input capability!");
        return 0;
    }

    int i;
    for ( i = 0; udi[i]; i++ )
    {
         printf("Device with input capability and udi=%s",udi[i]);
         /* Do something with it */
    }
    /* Free the string array */
    libhal_free_string_array(udi);
```

HAL callbacks

What about if we want to monitor a specific key on a given device, sure we are not going to ask HAL about this key every second this is very expensive to, so the HAL API offers callbacks function to watch system hardware.

Using HAL callbacks, we can register a callback function to be called when a key on a device changes. With callbacks we have the possibility to monitor one or more devices, or all the devices.

To add a watch function on a specific udi we use:

```
/* Adding a watch here will invoke the device_property_changed callback
   in case any property on this device changed */
dbus_bool_t libhal_device_add_property_watch (LibHalContext *ctx,
                                              const char *udi,
                                              DBusError *error);
/* To remove an already added watch function on a device */
dbus_bool_t libhal_device_remove_property_watch (LibHalContext *ctx,
                                                 const char *udi,
                                                 DBusError *error)
```

Here are common general callback functions, self explanatory:

```
dbus_bool_t    libhal_ctx_set_device_added             (LibHalContext *ctx,

                                                        LibHalDeviceAdded
callback);
dbus_bool_t    libhal_ctx_set_device_removed           (LibHalContext *ctx,
                                                        LibHalDeviceRemoved
callback);
dbus_bool_t    libhal_ctx_set_device_new_capability    (LibHalContext *ctx,
                                                        LibHalDeviceNewCapability
callback);
dbus_bool_t    libhal_ctx_set_device_lost_capability   (LibHalContext *ctx,
                                                        LibHalDeviceLostCapability
callback);
dbus_bool_t    libhal_ctx_set_device_property_modified (LibHalContext *ctx,
                                                        LibHalDevicePropertyModifi
ed callback);
```

We will use the glib main loop and integrate it with DBus connection using dbus-glib, see DBus tutorial

hal-callback.c

```
#include <glib.h>

#include <dbus/dbus-glib-lowlevel.h>
#include <dbus/dbus.h>
#include <hal/libhal.h>

static void
handle_device_removed(LibHalContext *ctx, const char *udi)
{
    printf("Device with udi=%s is removed\n", udi);
}

static void
handle_device_added(LibHalContext *ctx, const char *udi)
{
    printf("Device with udi=%s is added\n", udi);
}

int main()
{
    /* ... same as the above example */
```

```
    GMainLoop *loop;
    loop = g_main_loop_new(NULL,FALSE);

    dbus_connection_setup_with_g_main(connection,NULL);

    libhal_ctx_set_device_added(ctx, handle_device_added);
    libhal_ctx_set_device_removed(ctx, handle_device_removed);

    g_main_loop_run(loop);
}
```

Optionally before adding a callback function one can set data attached with HAL context using

```
libhal_ctx_set_user_data(ctx, pointer_to_the_data);
```

And find the data later

```
    user_data = libhal_ctx_get_user_data(ctx);
```

Compile with the following command:

```
gcc `pkg-config --libs --cflags dbus-1 hal dbus-glib-1 glib-2.0` callback-
example.c -o callback-example
```

Invoking HAL method:

In this section we will learn how to invoke an HAL methods, could be a DBus example, but since we are going to invoke an HAL method i preferred to have it here, to see the available HAL method on the device "computer" we can run the following command:

```
dbus-send --system --print-reply --dest=org.freedesktop.Hal \

            /org/freedesktop/Hal/devices/computer           \
            org.freedesktop.DBus.Introspectable.Introspect
```

First example is how to to ask HAL to put the system in hibernate state.

hibernate.c

```
#include <stdio.h>

#include <dbus/dbus.h>

#define HAL_DBUS_SERVICE             "org.freedesktop.Hal"
#define HAL_ROOT_COMPUTER            "/org/freedesktop/Hal/devices/computer"
#define HAL_DBUS_INTERFACE_POWER
"org.freedesktop.Hal.Device.SystemPowerManagement"

int main()
{
    DBusConnection *connection;
    DBusMessage *mess,*reply;
    DBusError error;
    int exit_code;
```

```
    dbus_error_init(&error);
    connection = dbus_bus_get(DBUS_BUS_SYSTEM, &error);

    if ( dbus_error_is_set(&error) )
    {
        printf("Unable to connect to the daemon bus: %s",error.message);
        dbus_error_free(&error);
        return 1;
    }

    /* Creating the DBus message that we want to send to HAL with Hibernate method
*/
    mess = dbus_message_new_method_call(HAL_DBUS_SERVICE,
                                        HAL_ROOT_COMPUTER,
                                        HAL_DBUS_INTERFACE_POWER,
                                        "Hibernate");
    if (!mess)
    {
        printf("Out of memory");
        return 1;
    }
    /* Send the message to HAL */
    reply = dbus_connection_send_with_reply_and_block(connection,
                                                      mess,
                                                      -1,            /* Default
DBus timeout */
                                                      &error);
    /* Free the message */
    dbus_message_unref(mess);

    if ( dbus_error_is_set(&error) )
    {
        /* Note here that not all the error raised by DBus should be taken into
consideration
           A real application should filter the errors, want more detail, mail me
*/
        printf("Error occured while trying to hibernate: %s", error.message);
        dbus_error_free(&error);
        return 1;
    }

    if ( !reply )
    {
        /* Also here this error could be ignored on a real application */
        printf("Message hibernate didn't get a reply");
        return 1;
    }

    /* If we have received a reply then there is a return code from the util that
HAL executed */
    switch(dbus_message_get_type(reply)) {

    case DBUS_MESSAGE_TYPE_METHOD_RETURN:
        dbus_message_get_args(reply,NULL,
                              DBUS_TYPE_INT32,
                              &exit_code,
                              DBUS_TYPE_INVALID);
```

```
        dbus_message_unref(reply);
        /* If exit_code is 0 so we are sure that we successfully hibernate the
system */
        if ( exit_code == 0 )
        {
            return 0;
        }
        if ( exit_code > 1 )
        {
            printf("System failed to hibernate");
            return 1;
        }
        break;
    case DBUS_MESSAGE_TYPE_ERROR:
        dbus_message_unref(reply);
        printf("Error occured while trying to hibernate");
        return 1;
    default:
        return 0;
    }

    dbus_connection_unref(connection);
    return 0;
}
```

Compile the above example as usual with dbus-1 argument for pkg-config, but don't run it unless you are sure that hibernate works on your system.

Using HAL libhal-storage:

In this section, we will write some code on the "handle_device_added" function from the hal-callback.c example in the previous page.

I suppose that the reader have a usb key or hdd, so this will be the device added, actually we are going just to read some information about the new device added, we are not going to mount/umount the device.

hal-storage-example.c

```
/* include this file instead of the previous hal/libhal.h */

#include <hal/libhal-storage.h>

static void
handle_device_added(LibHalContext *ctx, const char *udi)
{
    LibHalDrive *drive;
    dbus_bool_t is_storage;
```

```
    is_storage = libhal_device_query_capability(ctx, udi, "storage", NULL);

    if ( is_storage )
    {
        drive = libhal_drive_from_udi(ctx, udi);

        if ( libhal_drive_is_hotpluggable(drive) ||
libhal_drive_uses_removable_media(drive) )
        {
                printf("Storage device added %s model %s\n",
                        libhal_drive_get_device_file(drive),
                        libhal_drive_get_model(drive));
        }
        libhal_drive_free(drive);
    }
}
```

Compile the above example with the following command:

```
gcc `pkg-config --libs --cflags hal-storage dbus-1 dbus-glib-1 glib-2.0` \

hal-storage-example.c -o hal-storage-example
```