Introduction:

D-Bus is a message bus system, a simple way for applications to talk to one another, D-Bus supplies a system and a session daemons.

The system daemon is launched at the system startup level and used mainly for hardware events, while the session daemon is launched when the user login to a desktop environment and it is for use for desktop applications to connect to each other.

Note:The Dbus developer always recommend the usage of a DBus binding library, such as dbus-glib or dbus-qt, instead of using the DBus API directly, they said that the DBus API is not yet frozen and by using this API directly the programmer is signing up for some pain, in my opinion, in order to understand clearly any DBus binding libraries, it is a very good idea to dive into the DBus low level programming, keep in mind that what we are going to use here is the trivial part of the DBus API.

Before start doing any code, there is some terms that one should be familiar with:

DBus Connection:

DBusConnection is the structure to use for opening a connection to the daemon, either the system bus daemon by specifying DBUS_BUS_SYSTEM or to the session bus daemon using DBUS_BUS_SESSION.

DBus Message:

It is simply a message between two process, all the DBus intercommunication are done using DBusMessage, these messages can have the following types, method calls, method returns, signals, and errors. The DBusMessage structure can carry out parameters, by appending boolean integers, real numbers, string, arrays, ... to the message.

path:

Is the path of a remote Object, example /org/freedesktop/DBus.

Interface:

Is the interface on a given Object to talk with.

Signal:

It is a DBus message to make a signal emission.

Method Call:

It is a DBus message used to invoke a method on a remote object.

DBus Error:

The DBusError is the structure that holds the error code which occurs by calling a DBus method.

Getting a bus connection:

```
DBusConnection *connection;
DBusError error;

dbus_error_init(&error); /* Initialize the error structure */

connection = dbus_bus_get(DBUS_BUS_SESSION,&error); /* Or DBUS_BUS_SYSTEM */

if ( dbus_error_is_set(&error) )
{
    printf("Error connecting to the daemon bus: %s",error.message);
    dbus_error_free(&error);
}
```

For now, we have learned the basic concepts of DBus, next we will learn internal details of a DBus based application, by examples.

Example 1: Reserving a bus name

Just to get more familiar with DBus programming, in this example we will see how we can reserve a bus name for our small application. There is some restriction that DBus applies to the bus names, they are a valid UTF-8 string and must have at least one '.' which separates the elements name, each element must contains at least one character, example "org.freedesktop". , for a full list read the DBus specification section Bus names.

```
#include <stdio.h>
#include <dbus/dbus.h>

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

    char *name = "org.share.linux";

    dbus_error_init(&error);

    connection = dbus_bus_get(DBUS_BUS_SESSION, &error);

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

    dbus_bool_t ret = dbus_bus_name_has_owner(connection,name,&error);

    if ( dbus_error_is_set(&error) )
    {
```

```
        dbus_error_free(&error);
        printf("DBus Error: %s\n",error.message);
        return 1;
    }

    if ( ret == FALSE )
    {
        printf("Bus name %s doesn't have an owner, reserving it...\n",name);
        int request_name_reply =
        dbus_bus_request_name( connection,name, DBUS_NAME_FLAG_DO_NOT_QUEUE,
&error);
        if ( dbus_error_is_set(&error) )
        {
            dbus_error_free(&error);
            printf("Error requesting a bus name: %s\n",error.message);
            return 1;
        }

        if ( request_name_reply == DBUS_REQUEST_NAME_REPLY_PRIMARY_OWNER )
        {
            printf("Bus name %s Successfully reserved!\n",name);
            return 0;
        }
        else
        {
            printf("Failed to reserve name %s\n",name);
            return 1;
        }
    }
    else
    /*
    if ret of method dbus_bus_name_has_owner is TRUE, then this is useful for
    detecting if your application is already running and had reserved a bus name
    unless somebody stole this name from you, so better to choose a correct bus
name
    */
    {
        printf("%s is already reserved\n", name);
        return 1;
    }

    return 0;
}
```

We have used the DBUS_NAME_FLAG_DO_NOT_QUEUE flag, DBus will not queue us in case the bus name that we want to reserve is already used. For a full list of flags for dbus_bus_request_name and return code see the DBus API. A requested bus name can always be released using dbus_bus_release_name.

To Compile the above code, the DBus developement package should be installed, depending on your distro the name of this package may differ, but should be something like libdbus-dev (On a Slackware system all the package comes with their development files). Then you compile the code with the following command:

```
gcc `pkg-config --libs --cflags dbus-1` example1.c -o example1
```

pkg-config tries to find dbus-1.pc file, usually this file is located with others in /usr/lib/pkgconfig, and those kind of files contains information about the library(ies) to link.

Example 2: Connecting two desktop applications

In this example, we will use DBus to connect two desktop applications, one listen to DBus messages and the other send DBus messages, but before starting, the listener program should not just start and exit, it has to wait for events, so we have to find a way to organize the events sent to our program, a simple solution for this is to use the main event loop from glib, when using it we can keep our program in sleep mode until receiving events, another problem occurs is the fact that how we can integrate our bus connection with the glib main event loop, here comes dbus-glib, so our tiny program will depend also on dbus-glib for just one call, dbus_connection_setup_with_g_main, this call integrates the glib main loop and the DBus bus events.

A question raises here, if we want to use just DBus, how we can avoid the usage of its glib binding, the answer is not simple, first we have to write our own loop events, and integrate it with the bus events, a good start is to look at the DBus source as they have a helpful code in dbus/dbus-mainloop, but to simplify our job we will use dbus-glib.

listen.c In this program we will use dbus_bus_add_match(DbusConnection *,const char *rule, DBusError *) to add a match for the messages that we want to receive, the rule string has a specific format, see DBus match rule for full details.

```
#include <stdio.h>

#include <dbus/dbus.h>

#include <dbus/dbus-glib.h>
#include <glib.h>

static DBusHandlerResult
dbus_filter (DBusConnection *connection, DBusMessage *message, void *user_data)
{

    if ( dbus_message_is_signal(message,"org.share.linux","Customize" ) )
    {
        printf("Message cutomize received\n");
        return DBUS_HANDLER_RESULT_HANDLED;
    }

    if ( dbus_message_is_signal(message,"org.share.linux","Quit" ) )
    {
        printf("Message quit received\n");
        GMainLoop *loop = (GMainLoop*) user_data;
        g_main_loop_quit(loop);
        return DBUS_HANDLER_RESULT_HANDLED;
    }
```

```
    return DBUS_HANDLER_RESULT_NOT_YET_HANDLED;
}

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

    /* glib main loop */
    GMainLoop *loop;
    loop = g_main_loop_new(NULL,FALSE);

    dbus_error_init(&error);

    connection = dbus_bus_get(DBUS_BUS_SESSION, &error);

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

    dbus_bus_add_match (connection,
"type='signal',interface='org.share.linux'",NULL);
    dbus_connection_add_filter (connection, dbus_filter, loop, NULL);

    /* dbus-glib call */
    dbus_connection_setup_with_g_main(connection,NULL);

    /* run glib main loop */
    g_main_loop_run(loop);

    return 0;
}
```

send.c

```
#include <stdio.h>
#include <dbus/dbus.h>

static void
send_config(DBusConnection *connection)
{
    DBusMessage *message;
    message = dbus_message_new_signal ("/org/share/linux",
                                       "org.share.linux",
                                       "Config");
    /* Send the signal */
    dbus_connection_send (connection, message, NULL);
    dbus_message_unref (message);
}

static void
send_quit (DBusConnection *connection)
{
    DBusMessage *message;
    message = dbus_message_new_signal ("/org/share/linux",
                                       "org.share.linux",
```

```
                                          "Quit");
    /* Send the signal */
    dbus_connection_send (connection, message, NULL);
    dbus_message_unref (message);
}

int
main (int argc, char **argv)
{
    DBusConnection *connection;
    DBusError error;

    dbus_error_init (&error);
    connection = dbus_bus_get (DBUS_BUS_SESSION, &error);
    if (!connection)
    {
        printf ("Failed to connect to the D-BUS daemon: %s", error.message);
        dbus_error_free (&error);
        return 1;
    }

    if ( argc == 1 )
    {
        return 0;
    }

    int i;
    for ( i = 1; i < argc; i++)
    {
        if (!strcmp(argv[i],"-c") )
        {
            send_config(connection);
        }
        else if ( !strcmp(argv[i],"-q") )
        {
            send_quit(connection);
        }
    }
    return 0;
}
```

To compile run the following commands:

```
gcc `pkg-config --libs --cflags dbus-1 glib-2.0 dbus-glib-1` listen.c -o listen

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

Example 3: DBus services

The message bus can start applications (services) on behalf of other applications, the application asks DBus to start a service by its name, usually the name should be known such as org.freedesktop.TextEditor.

In order for DBus to find the executable corresponding to a particular name, the bus daemon looks for service description files which usually are installed in /usr/share/dbus-1/services and they have .service in their extension name (all linux distors that i know they use this prefix to install dbus services files), as an example of a service file.

DBus service file example:

```
[D-BUS Service]
Name=org.share.linux
Exec=path to the executable.
```

We will write two programs, one is the service that we want to start the other is the application that activates this service

share-linux-serivce-example.c

```
#include <stdio.h>
#include <dbus/dbus.h>

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

    dbus_error_init(&error);
    connection = dbus_bus_get(DBUS_BUS_STARTER, &error); /* DBUS_BUS_STARTER is
the bus that started us */

    /* Do something here to make sure that the application was successfully
started by DBus
     * Example could be something like
     * FILE *tmp;
     * tmp = fopen("/tmp/share-linux-service.result", "w");
     * fprintf(tmp,"share-linux service was started successfully");
     * fclose(tmp);
     * /

    /* After that you have the service up, so you can do whetever you like */

    dbus_connection_unref(connection);

    return 0;
}
```

```
Compile this example with dbus-1 argument for pkg-config, you need to install the
service file in /usr/share/dbus-1/service, name it org.share.linux and edit the
Exec path to where you have the service example binary.
```

start-service.c

```
#include <stdio.h>
#include <dbus/dbus.h>
```

```
int main()
{
    DBusConnection *connection;
    DBusError error;
    DBusMessage *message;

    const char *service_name = "org.share.linux";
    dbus_uint32_t flag; /* Currently this is not used by DBus, they say it is for
futur expansion*/
    dbus_bool_t result;

    dbus_error_init(&error);

    connection = dbus_bus_get(DBUS_BUS_SESSION, &error);

    if ( dbus_error_is_set(&error) )
    {
        printf("Error getting dbus connection: %s\n",error.message);
        dbus_error_free(&error);
        dbus_connection_unref(connection);
        return 0;
    }

    message = dbus_message_new_method_call("org.freedesktop.DBus",
                                           "/org/freedesktop/DBus",
                                           "org.freedesktop.DBus",
                                           "StartServiceByName");

    if ( !message )
    {
        printf("Error creating DBus message\n");
        dbus_connection_unref(connection);
        return 0;
    }

    dbus_message_set_no_reply(message, TRUE); /* We don't want to receive a reply
*/

    /* Append the argument to the message, must ends with DBUS_TYPE_UINT32 */
    dbus_message_append_args(message,
                             DBUS_TYPE_STRING,
                             &service_name,
                             DBUS_TYPE_UINT32,
                             &flag,
                             DBUS_TYPE_INVALID);

    result = dbus_connection_send(connection, message, NULL);

    if ( result == TRUE )
    {
        printf("Successfully activating the %s service\n",service_name);
    }
    else
    {
        printf("Failed to activate the %s service\n",service_name);
    }
    dbus_message_unref(message);
    dbus_connection_unref(connection);
```

```
    return 0;
}
```

Hopefully you found this small tutorial useful, as usual permission is granted to modify or copy the content of this document under the GNU General Public License GPL.