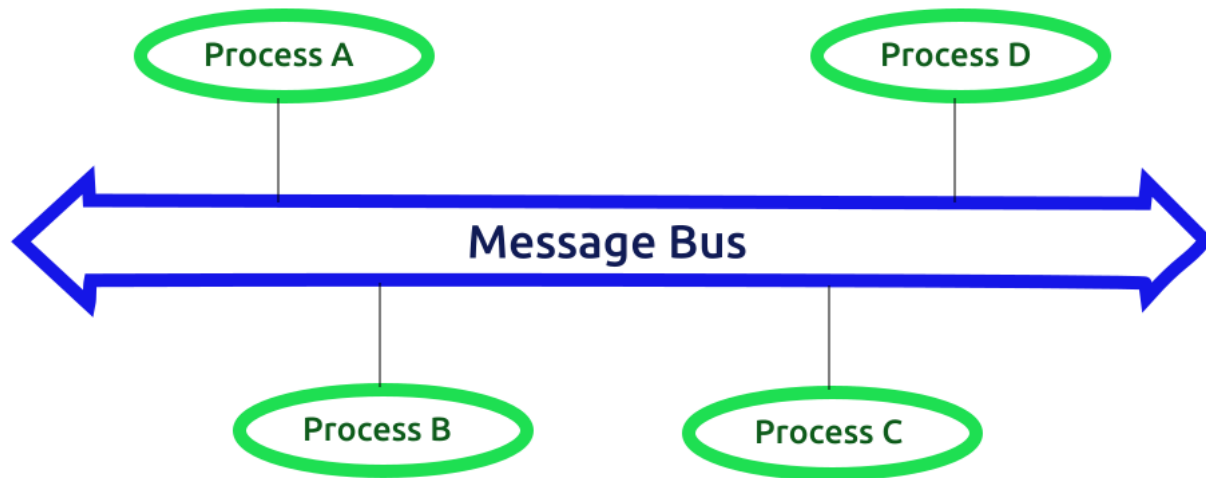


SoftPrayog ^(/)

D-Bus Tutorial

1.0 D-Bus

D-Bus is a mechanism for interprocess communication under Linux and other Unix-like systems. D-Bus has a layered architecture. At the lowest level is the D-Bus wire protocol described in the D-Bus Specification (<https://dbus.freedesktop.org/doc/dbus-specification.html>). The *libdbus* library is an implementation of the wire protocol. It provides the C Language interface for communication between two processes. Central to D-Bus is the concept of a message bus daemon which routes messages between processes. There are two message bus daemons. First, there is the **system D-BUS** daemon for communication between the kernel, system-wide services and the user. Second, there is the **session D-BUS** daemon, primarily intended for communication between processes for the desktop applications of the logged-in user.



Interprocess communication using D-Bus

libdbus is a low level API. There are high level bindings for multiple software frameworks, viz., GDBus for GLib, QtDBus for Qt, dbus-java and sd-bus, the last being a part of *systemd*.

2.0 D-Bus concepts

Message

A message is the unit of data transfer between processes. Messages are discrete as opposed to continuous stream of data transferred between processes by mechanisms like pipes. A message has a header, which identifies its sender, receiver and method or signal name, etc. and message body containing a data payload.

Message Types

There are four types of messages, `SIGNAL`, `METHOD_CALL`, `METHOD_RETURN` and `ERROR`.

A `SIGNAL` is a message that is broadcast by a process and can be received by other interested processes.

A `METHOD_CALL` message is a request by the sender for a particular operation on an object of the receiver. For example, the receiver may be a service with a singleton object. The sender could be a client, requesting the execution of a "method" by the server. The method call message has the name of the method to be executed and also the arguments required for execution. The receiver is required to execute the method and respond back to the sender with a `METHOD_RETURN` message containing the result(s) of the operation. Or, if there was an error, the receiver can respond with an `ERROR` message.

Message bus

A message bus is a daemon process which routes messages between other processes.

Service

A service is a daemon process that provides some utility in the system. A service is a server process which does work for the clients. A service has a singleton object.

Object

An object is an entity in a process, which does some work. An object is identified by a *path* name. A path is like a complete file name in the system. So, an object might have a path name like, /com/example/someserver. An object has members, which means methods and signals.

Interfaces

An interface is a group of functions. An object supports one or more interfaces. The interfaces supported by an object specify the members of that object.

Connection names

When an application connects with the D-Bus daemon, it is assigned a **unique connection name**. A unique connection name starts with the colon character ":".

An application may also ask for a *well-known name* to be assigned to a connection. This is of the form of a reverse domain name like, com.example.somenam.

For each connection name there is an application which is its primary owner. All messages sent to a name are delivered to the primary owner. Then, there is a queue of secondary owners. As soon as a primary owner relinquishes the name, the application at the top of the queue of aspiring secondary owners takes charge and becomes the new primary owner.

3.0 D-Bus Configuration

The D-Bus daemon configuration files are located in the `/usr/share/dbus-1` directory. The configuration files, `system.conf` and `session.conf` for the system bus and session bus respectively are also symbolically linked in the `/etc/dbus-1` directory. There are directives in `system.conf` and `session.conf` to include files in the `system.d` and `session.d` sub-directories, respectively.

4.0 Use Cases

D-Bus is used for interprocess communication between a server or service process and clients. There are two situations. First, there, is only one-way communication. The clients may inform the server of some events and the server makes a note of it. The server does not respond back to the client. Similarly, a service may broadcast some information and those (processes) which are interested, make a note of it. The clients do not respond back to the server. The second case is a full-fledged two-way communication. A client sends some information in a request message to the server. The server receives the message, processes the data and sends a reply message back to the client. The first case is accomplished using signals whereas the second is implemented using method calls. The example given below pertains to the second case.

5.0 A client-server example

The server in this example provides a simple addition service. The clients send two integers in a method call. The server adds them up and sends the answer as the reply. We will use the low-level *libdbus* interface from C programs.

5.1 Configuration files

The server configuration file, in this example, is `/etc/dbus-1/system.d/in.softprayog.add_server.conf`.

```
<!DOCTYPE busconfig PUBLIC
"-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>

  <policy user="root">
    <allow own="in.softprayog.add_server"/>
  </policy>

  <policy user="alice">
    <allow own="in.softprayog.add_server"/>
  </policy>

  <policy context="default">
    <allow send_interface="in.softprayog.dbus_example"/>
    <allow send_destination="in.softprayog.add_client"/>
  </policy>

</busconfig>
```

Similarly, the client configuration file, `/etc/dbus-1/system.d/in.softprayog.add_client.conf`, is:

```
<!DOCTYPE busconfig PUBLIC
"-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
```

```
"http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">
<busconfig>

  <policy user="root">
    <allow own="in.softprayog.add_client"/>
  </policy>

  <policy user="alice">
    <allow own="in.softprayog.add_client"/>
  </policy>

  <policy context="default">
    <allow send_interface="in.softprayog.dbus_example"/>
    <allow send_destination="in.softprayog.add_server"/>
  </policy>

</busconfig>
```

The server program is,

```
/*
 *
 *   add-server.c: server program, receives message,
 *                 adds numbers in message and
 *                 gives back result to client
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <ctype.h>
#include <dbus/dbus.h>

const char *const INTERFACE_NAME = "in.softprayog.dbus_example";
const char *const SERVER_BUS_NAME = "in.softprayog.add_server";
const char *const OBJECT_PATH_NAME = "/in/softprayog/adder";
const char *const METHOD_NAME = "add_numbers";

DBusError dbus_error;
void print_dbus_error (char *str);
bool isinteger (char *ptr);

int main (int argc, char **argv)
{
    DBusConnection *conn;
    int ret;

    dbus_error_init (&dbus_error);

    conn = dbus_bus_get (DBUS_BUS_SYSTEM, &dbus_error);

    if (dbus_error_is_set (&dbus_error))
        print_dbus_error ("dbus_bus_get");

    if (!conn)
        exit (1);

    // Get a well known name
    ret = dbus_bus_request_name (conn, SERVER_BUS_NAME, DBUS_NAME_FLAG_DO_NOT_QUEUE, &dbus_error);

    if (dbus_error_is_set (&dbus_error))
        print_dbus_error ("dbus_bus_get");

    if (ret != DBUS_REQUEST_NAME_REPLY_PRIMARY_OWNER) {
        fprintf (stderr, "DBus: not primary owner, ret = %d\n", ret);
        exit (1);
    }
}
```

```

// Handle request from clients
while (1) {
    // Block for msg from client
    if (!dbus_connection_read_write_dispatch (conn, -1)) {
        fprintf (stderr, "Not connected now.\n");
        exit (1);
    }

    DBusMessage *message;

    if (message = dbus_connection_pop_message (conn)) == NULL) {
        fprintf (stderr, "Did not get message\n");
        continue;
    }

    if (dbus_message_is_method_call (message, INTERFACE_NAME, METHOD_NAME)) {
        char *s;
        char *str1 = NULL, *str2 = NULL;
        const char space [4] = " \n\t";
        int i, j;
        bool error = false;

        if (dbus_message_get_args (message, &dbus_error, DBUS_TYPE_STRING, &s, DBUS_TYPE_INVALID)) {
            printf ("%s", s);
            // Validate received message
            str1 = strtok (s, space);
            if (str1)
                str2 = strtok (NULL, space);

            if (isinteger (str1))
                i = atoi (str1);
            else
                error = true;
            if (isinteger (str2))
                j = atoi (str2);
            else
                error = true;
            if (!error) {
                // send reply
                DBusMessage *reply;
                char answer [40];

                sprintf (answer, "Sum is %d", i + j);
                if ((reply = dbus_message_new_method_return (message)) == NULL) {
                    fprintf (stderr, "Error in dbus_message_new_method_return\n");
                    exit (1);
                }

                DBusMessageIter iter;
                dbus_message_iter_init_append (reply, &iter);
                char *ptr = answer;
                if (!dbus_message_iter_append_basic (&iter, DBUS_TYPE_STRING, &ptr)) {
                    fprintf (stderr, "Error in dbus_message_iter_append_basic\n");
                    exit (1);
                }

                if (!dbus_connection_send (conn, reply, NULL)) {
                    fprintf (stderr, "Error in dbus_connection_send\n");
                    exit (1);
                }

                dbus_connection_flush (conn);

                dbus_message_unref (reply);
            }
            else // There was an error
            {
                DBusMessage *dbus_error_msg;
                char error_msg [] = "Error in input";
                if ((dbus_error_msg = dbus_message_new_error (message, DBUS_ERROR_FAILED, error_msg)) == NULL) {
                    fprintf (stderr, "Error in dbus_message_new_error\n");
                    exit (1);
                }
            }
        }
    }
}

```

```

    }

    if (!dbus_connection_send (conn, dbus_error_msg, NULL)) {
        fprintf (stderr, "Error in dbus_connection_send\n");
        exit (1);
    }

    dbus_connection_flush (conn);

    dbus_message_unref (dbus_error_msg);
}
}
else
{
    print_dbus_error ("Error getting message");
}
}
}

return 0;
}

bool isinteger (char *ptr)
{
    if (*ptr == '+' || *ptr == '-')
        ptr++;

    while (*ptr) {
        if (!isdigit ((int) *ptr++))
            return false;
    }

    return true;
}

void print_dbus_error (char *str)
{
    fprintf (stderr, "%s: %s\n", str, dbus_error.message);
    dbus_error_free (&dbus_error);
}

```

And, the client program is

```

/*
 *
 *   add-client.c: client program, takes two numbers as input,
 *               sends to server for addition,
 *               gets result from server,
 *               prints the result on the screen
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>

#include <dbus/dbus.h>

const char *const INTERFACE_NAME = "in.softprayog.dbus_example";
const char *const SERVER_BUS_NAME = "in.softprayog.add_server";
const char *const CLIENT_BUS_NAME = "in.softprayog.add_client";
const char *const SERVER_OBJECT_PATH_NAME = "/in/softprayog/adder";
const char *const CLIENT_OBJECT_PATH_NAME = "/in/softprayog/add_client";
const char *const METHOD_NAME = "add_numbers";

```

```

DBusError dbus_error;
void print_dbus_error (char *str);

int main (int argc, char **argv)
{
    DBusConnection *conn;
    int ret;
    char input [80];

    dbus_error_init (&dbus_error);

    conn = dbus_bus_get (DBUS_BUS_SYSTEM, &dbus_error);

    if (dbus_error_is_set (&dbus_error))
        print_dbus_error ("dbus_bus_get");

    if (!conn)
        exit (1);

    printf ("Please type two numbers: ");
    while (fgets (input, 78, stdin) != NULL) {

        // Get a well known name
        while (1) {
            ret = dbus_bus_request_name (conn, CLIENT_BUS_NAME, 0, &dbus_error);

            if (ret == DBUS_REQUEST_NAME_REPLY_PRIMARY_OWNER)
                break;

            if (ret == DBUS_REQUEST_NAME_REPLY_IN_QUEUE) {
                fprintf (stderr, "Waiting for the bus ... \n");
                sleep (1);
                continue;
            }
            if (dbus_error_is_set (&dbus_error))
                print_dbus_error ("dbus_bus_get");
        }

        DBusMessage *request;

        if ((request = dbus_message_new_method_call (SERVER_BUS_NAME, SERVER_OBJECT_PATH_NAME,
                                                    INTERFACE_NAME, METHOD_NAME)) == NULL) {
            fprintf (stderr, "Error in dbus_message_new_method_call\n");
            exit (1);
        }

        DBusMessageIter iter;
        dbus_message_iter_init_append (request, &iter);
        char *ptr = input;
        if (!dbus_message_iter_append_basic (&iter, DBUS_TYPE_STRING, &ptr)) {
            fprintf (stderr, "Error in dbus_message_iter_append_basic\n");
            exit (1);
        }
        DBusPendingCall *pending_return;
        if (!dbus_connection_send_with_reply (conn, request, &pending_return, -1)) {
            fprintf (stderr, "Error in dbus_connection_send_with_reply\n");
            exit (1);
        }

        if (pending_return == NULL) {
            fprintf (stderr, "pending return is NULL");
            exit (1);
        }

        dbus_connection_flush (conn);

        dbus_message_unref (request);

        dbus_pending_call_block (pending_return);

        DBusMessage *reply;
        if ((reply = dbus_pending_call_steal_reply (pending_return)) == NULL) {
            fprintf (stderr, "Error in dbus_pending_call_steal_reply");
        }
    }
}

```

```

        exit (1);
    }

    dbus_pending_call_unref (pending_return);

    char *s;
    if (dbus_message_get_args (reply, &dbus_error, DBUS_TYPE_STRING, &s, DBUS_TYPE_INVALID)) {
        printf ("%s\n", s);
    }
    else
    {
        fprintf (stderr, "Did not get arguments in reply\n");
        exit (1);
    }
    dbus_message_unref (reply);

    if (dbus_bus_release_name (conn, CLIENT_BUS_NAME, &dbus_error) == -1) {
        fprintf (stderr, "Error in dbus_bus_release_name\n");
        exit (1);
    }

    printf ("Please type two numbers: ");
}

return 0;
}

void print_dbus_error (char *str)
{
    fprintf (stderr, "%s: %s\n", str, dbus_error.message);
    dbus_error_free (&dbus_error);
}

```

The makefile for building the software is,

```

#
# Makefile
#

CC=gcc
all: add-server add-client

%.o: %.c
    gcc -Wall -c `pkg-config --cflags dbus-1` $< -o $@

add-server: add-server.o
    $(CC) add-server.o -o add-server `pkg-config --libs dbus-1`

add-client: add-client.o
    $(CC) add-client.o -o add-client `pkg-config --libs dbus-1`

.PHONY: clean
clean:
    rm *.o add-server add-client

```

We can compile and run the server and client programs,

```

Terminal
alice@bagpipe: ~/src
$ make
gcc -Wall -c `pkg-config --cflags dbus-1` add-server.c -o add-server.o
gcc add-server.o -o add-server `pkg-config --libs dbus-1`
gcc -Wall -c `pkg-config --cflags dbus-1` add-client.c -o add-client.o
gcc add-client.o -o add-client `pkg-config --libs dbus-1`
$
$ ./add-server
3 4
44 23
1024 512
2048 2048
123 a
ab cf
123 456
-23 24

```

```

alice@bagpipe: ~
$ ./add-client
Please type two numbers: 3 4
Sum is 7
Please type two numbers: 2048 2048
Sum is 4096
Please type two numbers: 123 a
Error in input
Please type two numbers: -23 24
Sum is 1
Please type two numbers:

```

```

alice@bagpipe: ~
$ ./add-client
Please type two numbers: 44 23
Sum is 67
Please type two numbers: 1024 512
Sum is 1536
Please type two numbers: ab cf
Error in input
Please type two numbers: 123 456
Sum is 579
Please type two numbers:

```

6.0 REFERENCES

- D-Bus Specification (<https://dbus.freedesktop.org/doc/dbus-specification.html>)
- D-Bus low-level public API (https://dbus.freedesktop.org/doc/api/html/group__DBus.html)
- dbus-daemon (<https://dbus.freedesktop.org/doc/dbus-daemon.1.html>)

Share 1

Tweet

G+ Share 2

Share

0 Comments **SoftPrayog** Login ▾ Recommend  Share

Sort by Best ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Be the first to comment.

ALSO ON SOFTPRAYOG

ps command usage examples in Linux

1 comment • 2 years ago•

**Nikhil** — Hi, The ps command explanation is really helpful and can be used on Linux environment where different processes are running and can ...**Error while loading shared libraries: libz.so.1**





1 comment • 2 years ago•

**user1234** — thank you**How to redirect non-www URLs to www in Varnish**

1 comment • 2 years ago•

**Kojo** — Excellent. You made my day, thank you ! It is so common to handle this with the backend, but then it might cause no backend connection or ...**Random Quote**

3 comments • 2 years ago•

**Janice** — This is a great app for people like me who who use quotes daily for many reasons especially when I need a good quote to keep my head up ... Subscribe  Add Disqus to your site  Add Disqus  Privacy**App**

- Random Quotes (<https://randomquotesapp.com>)

(<https://play.google.com/store/apps/details?id=in.softprayog.randomquote>)

Share 1

Tweet

 Share 2 Share**Related Posts**

- Interprocess communication using POSIX message queues in Linux (</programming/interprocess-communication-using-posix-message-queues-in-linux>)
- Interprocess communication using POSIX Shared Memory in Linux (</programming/interprocess-communication-using-posix-shared-memory-in-linux>)
- Interprocess communication using FIFOs in Linux (</programming/interprocess-communication-using-fifos-in-linux>)
- Interprocess communication using System V message queues in Linux (</programming/interprocess-communication-using-system-v-message-queues-in-linux>)
- Interprocess communication using pipes in Linux (</programming/interprocess-communication-using-pipes-in-linux>)

Recent Posts (/all-posts)

- Analyzing the NGINX web server log file (</troubleshooting/analyzing-the-nginx-web-server-log-file>)
- Signals in Linux (</programming/signals-in-linux>)
- Android adb install failure INSTALL_PARSE_FAILED_NO_CERTIFICATES (</troubleshooting/android-adb-install-failure-install-parse-failed-no-certificates>)
- nginx drupal sitemap 404 not found (</troubleshooting/nginx-drupal-sitemap-404-not-found>)
- systemd - System and service manager in Linux (</tutorials/systemd-system-and-service-manager-in-linux>)

 (</rss.xml>)