# Maemo Diablo Source code for the libdbus example

Training Material

February 9, 2009

# Contents

1 Source code for the libdbus example 2
1.1 libdbus-example/dbus-example.c . . . . . . . . . . . . . . . . . . . 2
1.2 libdbus-example/Makefile . . . . . . . . . . . . . . . . . . . . . . 6

# Chapter 1

# Source code for the libdbus example

## 1.1 libdbus-example/dbus-example.c

```
/**
 * A simple D-Bus RPC sending example.
 *
 * This maemo code example is licensed under a MIT-style license,
 * that can be found in the file called "License" in the same
 * directory as this file.
 * Copyright (c) 2007-2008 Nokia Corporation. All rights reserved.
 *
 * The program will:
 * 1) Connect to the D-Bus Session bus
 * 2) Send one RPC method call (one that will cause a Note dialog to
 *    be popped up for user).
 * 3) Quit
 */

/**
 * Uses the low-level libdbus which shouldn't be used directly.
 * As the D-Bus API reference puts it "If you use this low-level API
 * directly, you're signing up for some pain".
 */

#include <dbus/dbus.h> /* Pull in all of D-Bus headers. */
#include <stdio.h>     /* printf, fprintf, stderr */
#include <stdlib.h>    /* EXIT_FAILURE, EXIT_SUCCESS */
#include <assert.h>    /* assert */

/* Symbolic defines for the D-Bus well-known name, interface, object
   path and method name that we're going to use. */

#define SYSNOTE_NAME  "org.freedesktop.Notifications"
#define SYSNOTE_OPATH "/org/freedesktop/Notifications"
#define SYSNOTE_IFACE "org.freedesktop.Notifications"
#define SYSNOTE_NOTE  "SystemNoteDialog"

/**
 * Utility to terminate if given DBusError is set.
 * Will print out the message and error before terminating.
 *
 * If error is not set, will do nothing.
 *
 * NOTE: In real applications you should spend a moment or two
 *       thinking about the exit-paths from your application and
 *       whether you need to close/unreference all resources that you
 *       have allocated. In this program, we rely on the kernel to do
 *       all necessary cleanup (closing sockets, releasing memory),
 *       but in real life you need to be more careful.
 *
 *       One possible solution model to this is implemented in
 *       "flashlight", a simple program that is presented later.
 */
static void terminateOnError(const char* msg,
                             const DBusError* error) {

  assert(msg != NULL);
  assert(error != NULL);

  if (dbus_error_is_set(error)) {
    fprintf(stderr, msg);
    fprintf(stderr, "DBusError.name: %s\n", error->name);
    fprintf(stderr, "DBusError.message: %s\n", error->message);
    /* If the program wouldn't exit because of the error, freeing the
       DBusError needs to be done (with dbus_error_free(error)).
       NOTE:
         dbus_error_free(error) would only free the error if it was
         set, so it is safe to use even when you're unsure. */
    exit(EXIT_FAILURE);
  }
}

/**
 * The main program that demonstrates a simple "fire & forget" RPC
 * method invocation.
 */
int main(int argc, char** argv) {

  /* Structure representing the connection to a bus. */
  DBusConnection* bus = NULL;
  /* The method call message. */
  DBusMessage* msg = NULL;

  /* D-Bus will report problems and exceptions using the DBusError
     structure. We'll allocate one in stack (so that we don't need to
     free it explicitly. */
  DBusError error;

  /* Message to display. */
  const char* dispMsg = "Hello World!";
  /* Text to use for the acknowledgement button. "" means default. */
  const char* buttonText = "";
  /* Type of icon to use in the dialog (1 = OSSO_GN_ERROR). We could
     have just used the symbolic version here as well, but that would
     have required pulling the LibOSSO-header files. And this example
     must work without LibOSSO, so this is why a number is used. */
  int iconType = 1;

  /* Clean the error state. */
  dbus_error_init(&error);

  printf("Connecting to Session D-Bus\n");
```

```c
bus = dbus_bus_get(DBUS_BUS_SESSION, &error);
terminateOnError("Failed to open Session bus\n", &error);
assert(bus != NULL);

/* Normally one would just do the RPC call immediately without
   checking for name existence first. However, sometimes it's useful
   to check whether a specific name even exists on a platform on
   which you're planning to use D-Bus.

   In our case it acts as a reminder to run this program using the
   run-standalone.sh script when running in the SDK.

   The existence check is not necessary if the recipient is
   startable/activateable by D-Bus. In that case, if the recipient
   is not already running, the D-Bus daemon will start the
   recipient (a process that has been registered for that
   well-known name) and then passes the message to it. This
   automatic starting mechanism will avoid the race condition
   discussed below and also makes sure that only one instance of
   the service is running at any given time. */
printf("Checking whether the target name exists ("
       SYSNOTE_NAME ")\n");
if (!dbus_bus_name_has_owner(bus, SYSNOTE_NAME, &error)) {
  fprintf(stderr, "Name has no owner on the bus!\n");
  return EXIT_FAILURE;
}
terminateOnError("Failed to check for name ownership\n", &error);
/* Someone on the Session bus owns the name. So we can proceed in
   relative safety. There is a chance of a race. If the name owner
   decides to drop out from the bus just after we check that it is
   owned, our RPC call (below) will fail anyway. */

/* Construct a DBusMessage that represents a method call.
   Parameters will be added later. The internal type of the message
   will be DBUS_MESSAGE_TYPE_METHOD_CALL. */
printf("Creating a message object\n");
msg = dbus_message_new_method_call(SYSNOTE_NAME, /* destination */
                                   SYSNOTE_OPATH,  /* obj. path */
                                   SYSNOTE_IFACE,  /* interface */
                                   SYSNOTE_NOTE); /* method str */
if (msg == NULL) {
  fprintf(stderr, "Ran out of memory when creating a message\n");
  exit(EXIT_FAILURE);
}

/* Set the "no-reply-wanted" flag into the message. This also means
   that we cannot reliably know whether the message was delivered or
   not, but since we don't have reply message handling here, it
   doesn't matter. The "no-reply" is a potential flag for the remote
   end so that they know that they don't need to respond to us.

   If the no-reply flag is set, the D-Bus daemon makes sure that the
   possible reply is discarded and not sent to us. */
dbus_message_set_no_reply(msg, TRUE);

/* Add the arguments to the message. For the Note dialog, we need
   three arguments:
     arg0: (STRING) "message to display, in UTF-8"
     arg1: (UINT32) type of dialog to display. We will use 1.
                    (libosso.h/OSSO_GN_ERROR).
     arg2: (STRING) "text to use for the ack button". "" means
                    default text (OK in our case).
     When listing the arguments, the type needs to be specified first
     (by using the libdbus constants) and then a pointer to the
     argument content needs to be given.

     NOTE: It is always a pointer to the argument value, not the value
           itself!

     We terminate the list with DBUS_TYPE_INVALID. */
  printf("Appending arguments to the message\n");
  if (!dbus_message_append_args(msg,
                                DBUS_TYPE_STRING, &dispMsg,
                                DBUS_TYPE_UINT32, &iconType,
                                DBUS_TYPE_STRING, &buttonText,
                                DBUS_TYPE_INVALID)) {
    fprintf(stderr, "Ran out of memory while constructing args\n");
    exit(EXIT_FAILURE);
  }

  printf("Adding message to client's send-queue\n");
  /* We could also get a serial number (dbus_uint32_t) for the message
     so that we could correlate responses to sent messages later. In
     our case there won't be a response anyway, so we don't care about
     the serial, so we pass a NULL as the last parameter. */
  if (!dbus_connection_send(bus, msg, NULL)) {
    fprintf(stderr, "Ran out of memory while queueing message\n");
    exit(EXIT_FAILURE);
  }

  printf("Waiting for send-queue to be sent out\n");
  dbus_connection_flush(bus);

  printf("Queue is now empty\n");

  /* Now we could in theory wait for exceptions on the bus, but since
     this is only a simple D-Bus example, we'll skip that. */

  printf("Cleaning up\n");

  /* Free up the allocated message. Most D-Bus objects have internal
     reference count and sharing possibility, so _unref() functions
     are quite common. */
  dbus_message_unref(msg);
  msg = NULL;

  /* Free-up the connection. libdbus attempts to share existing
     connections for the same client, so instead of closing down a
     connection object, it is unreferenced. The D-Bus library will
     keep an internal reference to each shared connection, to
     prevent accidental closing of shared connections before the
     library is finalized. */
  dbus_connection_unref(bus);
  bus = NULL;

  printf("Quitting (success)\n");

  return EXIT_SUCCESS;
}
```

Listing 1.1: libdbus-example/dbus-example.c

## 1.2 libdbus-example/Makefile

```
#
# Simple Makefile for the libdbus example that will send the "Display
# Note dialog" RPC message. You need to run the example in the SDK or
# a compatible device.
#

# Define a list of pkg-config packages we want to use
pkg_packages := dbus-glib-1

PKG_CFLAGS  := $(shell pkg-config --cflags $(pkg_packages))
PKG_LDFLAGS := $(shell pkg-config --libs $(pkg_packages))
# Additional flags for the compiler:
#    -g : Add debugging symbols
# -Wall : Enable most gcc warnings
ADD_CFLAGS := -g -Wall

# Combine user supplied, additional, and pkg-config flags
CFLAGS  := $(PKG_CFLAGS) $(ADD_CFLAGS) $(CFLAGS)
LDFLAGS := $(PKG_LDFLAGS) $(LDFLAGS)

targets = dbus-example

.PHONY: all clean
all: $(targets)

dbus-example: dbus-example.c
  $(CC) $(CFLAGS) $< -o $@ $(LDFLAGS)

clean:
  $(RM) $(targets)
```

Listing 1.2: libdbus-example/Makefile