Maemo Diablo Source code for the LibOSSO RPC examples

Training Material

February 9, 2009

Contents

Chapter 1

Source code for the LibOSSO RPC examples

1.1 libosso-example-sync/libosso-rpc-sync.c

```
/**
 * A program that will issue a "system_note_dialog" RPC method call.
 *
 * 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.
 *
 * In function does the same as dbus-example.c, but this time
 * utilizing LibOSSO _rpc-functions (synchronously).
 *
 * It would be unfair to compare this program directly against the
 * dbus-example.c, since this program contains additional
 * functionality. In general using the LibOSSO _rpc functions will
 * lead to more compact code, but there are cases when you need to go
 * beyond what LibOSSO provides (and using the GLib/D-Bus interface is
 * the proper way to go in that case).
 *
 * Please run with run-standalone.sh (LibOSSO initialization will fail
 * otherwise).
 */

#include <glib.h>

#include <libosso.h>

#include <stdlib.h> /* EXIT_SUCCESS */

/* Symbolic defines for the D-Bus well-known name, interface, object
   path and method name that we're going to use. Same as before.
   These are defined in osso-internal.h but we cannot use it here. */

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

```
/**
 * Utility to return a pointer to a statically allocated string giving
 * the textural representation of LibOSSO errors. Has no internal
 * state (safe to use from threads).
 *
 * LibOSSO does not come with a function for this, so we define one
 * ourselves.
 */
static const gchar* ossoErrorStr(osso_return_t errCode) {

  switch (errCode) {
    case OSSO_OK:
      return "No error (OSSO_OK)";
    case OSSO_ERROR:
      return "Some kind of error occurred (OSSO_ERROR)";
    case OSSO_INVALID:
      return "At least one parameter is invalid (OSSO_INVALID)";
    case OSSO_RPC_ERROR:
      return "Osso RPC method returned an error (OSSO_RPC_ERROR)";
    case OSSO_ERROR_NAME:
      return "(undocumented error) (OSSO_ERROR_NAME)";
    case OSSO_ERROR_NO_STATE:
      return "No state file found to read (OSSO_ERROR_NO_STATE)";
    case OSSO_ERROR_STATE_SIZE:
      return "Size of state file unexpected (OSSO_ERROR_STATE_SIZE)";
    default:
      return "Unknown/Undefined";
  }
}

/**
 * Utility to print out the type and content of given osso_rpc_t.
 * It also demonstrates the types available when using LibOSSO for
 * the RPC. Most simple types are available, but arrays are not
 * (unfortunately).
 */
static void printOssoValue(const osso_rpc_t* val) {

  g_assert(val != NULL);

  switch (val->type) {
    case DBUS_TYPE_BOOLEAN:
      g_print("boolean:%s", (val->value.b == TRUE)?"TRUE":"FALSE");
      break;
    case DBUS_TYPE_DOUBLE:
      g_print("double:%.3f", val->value.d);
      break;
    case DBUS_TYPE_INT32:
      g_print("int32:%d", val->value.i);
      break;
    case DBUS_TYPE_UINT32:
      g_print("uint32:%u", val->value.u);
      break;
    case DBUS_TYPE_STRING:
      g_print("string:'%s'", val->value.s);
      break;
    case DBUS_TYPE_INVALID:
      g_print("invalid/void");
      break;
    default:
      g_print("unknown(type=%d)", val->type);
      break;
```

```
  }
}

/**
 * Do the RPC call.
 *
 * Note that this function will block until the method call either
 * succeeds, or fails. If the method call would take a long time to
 * run, this would block the GUI of the program (which we don't have).
 *
 * Needs the LibOSSO state to do the launch.
 */
static void runRPC(osso_context_t* ctx) {

  /* Message to display. */
  const char* dispMsg = "Hello SystemNote!\nVia LibOSSO/sync.";
  /* Icon type to use. */
  gint iconType = OSSO_GN_ERROR;
  /* Button label text to use, "" means leaving the defaults. */
  const char* labelText = "";

  /* Will hold the result from the RPC invocation function. */
  osso_return_t result;
  /* Will hold the result of the method call (or error). */
  osso_rpc_t methodResult = {};

  g_print("runRPC called\n");

  g_assert(ctx != NULL);

  /* Compared to the libdbus functions, LibOSSO provides conveniently
     a function that will do the dispatch and also allows us to pass
     the arguments all with one call.

     The arguments for the "SystemNoteDialog" are the same as in
     dbus-example.c (since it is the same service). You might also
     notice that even if LibOSSO provides some convenience, it does
     not completely isolate us from libdbus. We still supply the
     argument types using D-Bus constants.

     NOTE Do not pass the argument values by pointers as with libdbus,
          instead pass them by value (as below). */
  result = osso_rpc_run(ctx,
                        SYSNOTE_NAME,      /* well-known name */
                        SYSNOTE_OPATH,         /* object path */
                        SYSNOTE_IFACE,           /* interface */
                        SYSNOTE_NOTE,          /* method name */
                        &methodResult, /* method return value */
                        /* The arguments for the RPC. The types
                           are unchanged, but instead of passing
                           them via pointers, they're passed by
                           "value" instead. */
                        DBUS_TYPE_STRING, dispMsg,
                        DBUS_TYPE_UINT32, iconType,
                        DBUS_TYPE_STRING, labelText,
                        DBUS_TYPE_INVALID);
  /* Check whether launching the RPC succeeded. */
  if (result != OSSO_OK) {
    g_error("Error launching the RPC (%s)\n",
            ossoErrorStr(result));
    /* We also terminate right away since there's nothing to do. */
  }
```

```
  g_print("RPC launched successfully\n");

  /* Now decode the return data from the method call.
     NOTE: If there is an error during RPC delivery, the return value
           will be a string. It is not possible to differentiate that
           condition from an RPC call that returns a string.

     If a method returns "void", the type-field in the methodResult
     will be set to DBUS_TYPE_INVALID. This is not an error. */
  g_print("Method returns: ");
  printOssoValue(&methodResult);
  g_print("\n");

  g_print("runRPC ending\n");
}

int main(int argc, char** argv) {

  /* The LibOSSO context that we need to do RPC. */
  osso_context_t* ossoContext = NULL;

  g_print("Initializing LibOSSO\n");
  /* The program name for registration is communicated from the
     Makefile via a -D preprocessor directive. Since it doesn't
     contain any dots in it, a prefix of "com.nokia." will be added
     to it internally within osso_initialize(). */
  ossoContext = osso_initialize(ProgName, "1.0", FALSE, NULL);
  if (ossoContext == NULL) {
    g_error("Failed to initialize LibOSSO\n");
  }

  g_print("Invoking the method call\n");
  runRPC(ossoContext);

  g_print("Shutting down LibOSSO\n");
  /* Deinitialize LibOSSO. The function doesn't return status code so
     we cannot know whether it succeeded or failed. We assume that it
     always succeeds. */
  osso_deinitialize(ossoContext);
  ossoContext = NULL;

  g_print("Quitting\n");
  return EXIT_SUCCESS;
}
```

Listing 1.1: libosso-example-sync/libosso-rpc-sync.c

1.2 libosso-example-sync/Makefile

```
#
# Simple Makefile to build the LibOSSO/Async-RPC example
#

# define a list of pkg-config packages we want to use
pkg_packages := glib-2.0 libosso

PKG_CFLAGS  := $(shell pkg-config --cflags $(pkg_packages))
PKG_LDFLAGS := $(shell pkg-config --libs $(pkg_packages))
# add warnings and debugging info CFLAGS-variable
```

```
ADD_CFLAGS += -g -Wall

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

targets = libosso-rpc-sync

all: $(targets)

libosso-rpc-sync: libosso-rpc-sync.c
  $(CC) $(CFLAGS) -DProgName=\"LibOSSOExample\" \
    $< -o $@ $(LDFLAGS)

.PHONY: clean all
clean:
  $(RM) $(targets)
```

Listing 1.2: libosso-example-sync/Makefile

1.3 libosso-example-async/libosso-rpc-async.c

```
/**
 * A program that will issue a "system_note_dialog" RPC method call.
 *
 * 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.
 *
 * In function does the same as dbus-example.c, but this time
 * utilizing LibOSSO _rpc-functions and demonstrating how to integrate
 * asynchronous calls with LibOSSO.
 *
 * It would be unfair to compare this program directly against the
 * dbus-example.c, since this program contains additional
 * functionality. In general using the LibOSSO _rpc functions will
 * lead to more compact code, but there are cases when you need to go
 * beyond what LibOSSO provides (and using the GLib/D-Bus interface is
 * the proper way to go in that case).
 *
 * Please run with run-standalone.sh (LibOSSO initialization will fail
 * otherwise).
 */

#include <glib.h>
#include <libosso.h>
#include <stdlib.h> /* EXIT_SUCCESS */

/* Symbolic defines for the D-Bus well-known name, interface, object
   path and method name that we're going to use. Same as before.
   These are defined in osso-internal.h but we cannot use it here. */

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

/**
```

```
 * Small application state so that we can pass both LibOSSO context
 * and the mainloop around to the callbacks.
 */
typedef struct {
  /* A mainloop object that will "drive" our example. */
  GMainLoop* mainloop;
  /* The LibOSSO context which we use to do RPC. */
  osso_context_t* ossoContext;
} ApplicationState;

/**
 * Utility to return a pointer to a statically allocated string giving
 * the textural representation of LibOSSO errors. Has no internal
 * state (safe to use from threads).
 *
 * LibOSSO does not come with a function for this, so we define one
 * ourselves.
 */
static const gchar* ossoErrorStr(osso_return_t errCode) {

  switch (errCode) {
    case OSSO_OK:
      return "No error (OSSO_OK)";
    case OSSO_ERROR:
      return "Some kind of error occurred (OSSO_ERROR)";
    case OSSO_INVALID:
      return "At least one parameter is invalid (OSSO_INVALID)";
    case OSSO_RPC_ERROR:
      return "Osso RPC method returned an error (OSSO_RPC_ERROR)";
    case OSSO_ERROR_NAME:
      return "(undocumented error) (OSSO_ERROR_NAME)";
    case OSSO_ERROR_NO_STATE:
      return "No state file found to read (OSSO_ERROR_NO_STATE)";
    case OSSO_ERROR_STATE_SIZE:
      return "Size of state file unexpected (OSSO_ERROR_STATE_SIZE)";
    default:
      return "Unknown/Undefined";
  }
}

/**
 * Utility to print out the type and content of given osso_rpc_t.
 * It also demonstrates the types available when using LibOSSO for
 * the RPC. Most simple types are available, but arrays are not
 * (unfortunately).
 */
static void printOssoValue(const osso_rpc_t* val) {

  g_assert(val != NULL);

  switch (val->type) {
    case DBUS_TYPE_BOOLEAN:
      g_print("boolean:%s", (val->value.b == TRUE)?"TRUE":"FALSE");
      break;
    case DBUS_TYPE_DOUBLE:
      g_print("double:%.3f", val->value.d);
      break;
    case DBUS_TYPE_INT32:
      g_print("int32:%d", val->value.i);
      break;
    case DBUS_TYPE_UINT32:
      g_print("uint32:%u", val->value.u);
```

```
      break;
    case DBUS_TYPE_STRING:
      g_print("string:'%s'", val->value.s);
      break;
    case DBUS_TYPE_INVALID:
      g_print("invalid/void");
      break;
    default:
      g_print("unknown(type=%d)", val->type);
      break;
  }
}

/**
 * Will be called from LibOSSO when the RPC return data is available.
 * Will print out the result, and return. Note that it must not free
 * the value, since it does not own it.
 *
 * The prototype (for reference) must be osso_rpc_async_f().
 *
 * The parameters for the callback are the D-Bus interface and method
 * names (note that object path and well-known name are NOT
 * communicated). The idea is that you can then reuse the same
 * callback to process completions from multiple simple RPC calls.
 */
static void rpcCompletedCallback(const gchar* interface,
                                 const gchar* method,
                                 osso_rpc_t* retVal,
                                 gpointer userData) {

  ApplicationState* state = (ApplicationState*)userData;

  g_print("rpcCompletedCallback called\n");

  g_assert(interface != NULL);
  g_assert(method != NULL);
  g_assert(retVal != NULL);
  g_assert(state != NULL);

  g_print(" interface: %s\n", interface);
  g_print(" method: %s\n", method);
  /* NOTE If there is an error in the RPC delivery, the return value
          will be a string. This is unfortunate if your RPC call is
          supposed to return a string as well, since it is not
          possible to differentiate between the two cases.

          If a method returns "void", the type-field in the retVal
          will be set to DBUS_TYPE_INVALID (it's not an error). */
  g_print(" result: ");
  printOssoValue(retVal);
  g_print("\n");

  /* Tell the main loop to terminate. */
  g_main_loop_quit(state->mainloop);

  g_print("rpcCompletedCallback done\n");
}

/**
 * We launch the RPC call from within a timer callback in order to
 * make sure that a mainloop object will be running when the RPC will
 * return (to avoid a nasty race condition).
```

```
 *
 * So, in essence this is a one-shot timer callback.
 *
 * In order to launch the RPC, it will need to get a valid LibOSSO
 * context (which is carried via the userData/application state
 * parameter).
 */
static gboolean launchRPC(gpointer userData) {

  ApplicationState* state = (ApplicationState*)userData;
  /* Message to display. */
  const char* dispMsg = "Hello SystemNote!\nVia LibOSSO/async.";
  /* Icon type to use. */
  gint iconType = OSSO_GN_ERROR;
  /* Button label text to use. */
  const char* labelText = "Execute!";

  /* Will hold the result from the RPC launch call. */
  osso_return_t result;

  g_print("launchRPC called\n");

  g_assert(state != NULL);

  /* Compared to the libdbus functions, LibOSSO provides conveniently
     a function that will do the dispatch, registration of callbacks
     and allow argument supplying in one call.

     The arguments for the "system_note_dialog" are the same as in
     dbus-example.c (since it is the same service). You might also
     notice that even if LibOSSO provides some convenience, it does
     not completely isolate us from libdbus. We still supply the
     argument types using D-Bus constants.

     NOTE Do not pass the argument values by pointers as with libdbus,
          instead pass them by value (as below). */

  /* The only difference compared to the synchronous version is the
     addition of the callback function parameter, and the user-data
     parameter for data that will be passed to the callback. */
  result = osso_rpc_async_run(state->ossoContext,
                              SYSNOTE_NAME,     /* well-known name */
                              SYSNOTE_OPATH,        /* object path */
                              SYSNOTE_IFACE,          /* interface */
                              SYSNOTE_NOTE,         /* method name */
                              rpcCompletedCallback,    /* async cb */
                              state,           /* user-data for cb */
                              /* The arguments for the RPC. */
                              DBUS_TYPE_STRING, dispMsg,
                              DBUS_TYPE_UINT32, iconType,
                              DBUS_TYPE_STRING, labelText,
                              DBUS_TYPE_INVALID);
  /* Check whether launching the RPC succeeded (we don't know the
     result from the RPC itself). */
  if (result != OSSO_OK) {
    g_error("Error launching the RPC (%s)\n",
            ossoErrorStr(result));
    /* We also terminate right away since there's nothing to do. */
  }
  g_print("RPC launched successfully\n");

  g_print("launchRPC ending\n");
```

```
  /* We only want to be called once, so ask the caller to remove this
     callback from the timer launch list by returning FALSE. */
  return FALSE;
}

int main(int argc, char** argv) {

  /* Keep the application state in main's stack. */
  ApplicationState state = {};
  /* Keeps the results from LibOSSO functions for decoding. */
  osso_return_t result;
  /* Default timeout for RPC calls in LibOSSO. */
  gint rpcTimeout;

  g_print("Initializing LibOSSO\n");
  state.ossoContext = osso_initialize(ProgName, "1.0", FALSE, NULL);
  if (state.ossoContext == NULL) {
    g_error("Failed to initialize LibOSSO\n");
  }

  /* Print out the default timeout value (which we don't change, but
     could, with osso_rpc_set_timeout()). */
  result = osso_rpc_get_timeout(state.ossoContext, &rpcTimeout);
  if (result != OSSO_OK) {
    g_error("Error getting default RPC timeout (%s)\n",
            ossoErrorStr(result));
  }
  /* Interestingly the timeout seems to be -1, but is something else
     (by default). -1 probably then means that "no timeout has been
     set". */
  g_print("Default RPC timeout is %d (units)\n", rpcTimeout);

  g_print("Creating a mainloop object\n");
  /* Create a GMainLoop with default context and initial condition of
     not running (FALSE). */
  state.mainloop = g_main_loop_new(NULL, FALSE);
  if (state.mainloop == NULL) {
    g_error("Failed to create a GMainLoop\n");
  }

  g_print("Adding timeout to launch the RPC in one second\n");
  /* This could be replaced by g_idle_add(cb, &state), in order to
     guarantee that the RPC would be launched only after the mainloop
     has started. We opt for a timeout here (for no particular
     reason). */
  g_timeout_add(1000, (GSourceFunc)launchRPC, &state);

  g_print("Starting mainloop processing\n");
  g_main_loop_run(state.mainloop);

  g_print("Out of mainloop, shutting down LibOSSO\n");
  /* Deinitialize LibOSSO. */
  osso_deinitialize(state.ossoContext);
  state.ossoContext = NULL;

  /* Free GMainLoop as well. */
  g_main_loop_unref(state.mainloop);
  state.mainloop = NULL;

  g_print("Quitting\n");
  return EXIT_SUCCESS;
```

```
}
```

Listing 1.3: libosso-example-async/libosso-rpc-async.c

1.4 libosso-example-async/Makefile

```
#
# Simple Makefile to build the LibOSSO/Async-RPC example
#

# Define a list of pkg-config packages we want to use
pkg_packages := glib-2.0 libosso

PKG_CFLAGS  := $(shell pkg-config --cflags $(pkg_packages))
PKG_LDFLAGS := $(shell pkg-config --libs $(pkg_packages))
# Add warnings and debugging info
ADD_CFLAGS += -g -Wall

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

targets = libosso-rpc-async

all: $(targets)

libosso-rpc-async: libosso-rpc-async.c
  $(CC) $(CFLAGS) -DProgName=\"LibOSSOExample\" \
    $< -o $@ $(LDFLAGS)

.PHONY: clean all
clean:
  $(RM) $(targets)
```

Listing 1.4: libosso-example-async/Makefile