CSC 195.01 2013S *Technologies for Mediascripting*

Class 08: Inter-Application Communication with D-Bus (3): Writing Clients

Held: Thursday, 14 March 2013

Summary: We consider the basic steps in writing a D-Bus client.

Related Pages:

- EBoard.

Notes:

- We'll spend a few minutes going over your homework.
- Homework: Write a client that takes info from the command line and calls your server to do some computation with that info.
- `GDBusProxy` is documented at https://developer.gnome.org/gio/2.28/GDBusProxy.html.

Overview:

- Client actions
- Setup
- Method calls
- Cleanup
- Going Beyond the basics
- Introspection

Client Tasks

- What kinds of things does a client have to do?
- Setup (almost always)
  - Connect to the bus
  - Connect to a particular object on a server
- Setup (often)
  - Query the bus for services
  - Query a service for its objects
  - Query an object for its properties, methods, and signals
- Main body
  - Send requests to the object and get responses
  - Listen for signals
- Cleanup
  - Close the connection

Setting Up The Client

- I find the `GDBusProxy` library the easiest way to deal with connecting to servers.
- There are a variety of ways to set up a proxy. I find the easiest to be `g_dbus_proxy_new_for_bus_sync`
    - It's synchronous, rather than aysnchronous, so I don't have to provide a callback and deal with all of that strange logic.
    - It expects a bus type, rather than a bus connection, so I don't have to worry about setting up the connection first. (You can see our server code for how to set up a connection.)
    - Here's the signature

```
GDBusProxy *g_dbus_proxy_new_for_bus_sync (GBusType bus_type,
                                           GDBusProxyFlags flags,
                                           GDBusInterfaceInfo *info,
                                           const gchar *name,
                                           const gchar *object_path,
                                           const gchar *interface_name,
                                           GCancellable *cancellable,
                                           GError **error);
```

    - And a sample call

```
g_dbus_proxy_new_for_bus_sync (G_BUS_TYPE_SESSION,
                               G_DBUS_PROXY_FLAGS_NONE,
                               NULL,
                               service,
                               object,
                               interface,
                               NULL,
                               errorp);
```

Calls

- The primary mechanism for calls is `g_dbus_proxy_call`. This procedure is asynchronous.
- There's also a synchronous alternative (which I tend to prefer during development): `g_dbus_proxy_call_sync`.
- Here's the signature.

```
GVariant *g_dbus_proxy_call_sync (GDBusProxy *proxy,
                                  const gchar *method_name,
                                  GVariant *parameters,
                                  GDBusCallFlags flags,
                                  gint timeout_msec,
                                  GCancellable *cancellable,
                                  GError **error);
```

- You get to build the parameters like you've done with the return value on your sample server.
- You get to extract the result like you extracted parameters in the server.

Cleanup

- Believe it or not, but there’s not a lot of cleanup to do. We tend to use `g_object_unref` when we’re done with a proxy and `g_variant_unref` when we’re done with a variant. That’s about it.
- (My sample code doesn’t do such a great job.)

Going Beyond the Basics

- Of course, we rarely want to write programs that just call specific functions on the server.
- What do we do instead? Often, we provide natural ways for something to “talk to” the server.
  - A simple user interface
  - A programmer’s interface (e.g., bridging languages)
  - ...

Introspecting Objects

- What if you don’t know a lot about an object. What can you find out?
- Every object supports the org.freedesktop.DBus.Introspectable.Introspect method, which returns XML to describe the object.
- Here’s the code I use to grab info and return it in an easy to proces form.

```
/**
 * Get information on a proxied object.
 */
static GDBusNodeInfo *
g_dbus_proxy_get_node_info (GDBusProxy *proxy)
{
  GError *error;                // Error returned by various functions.
  GVariant *response;           // The response from the proxy call.
  GDBusNodeInfo *info;          // Information on the node.
  const gchar *xml;             // XML code for the proxy interface.

  // Get the introspection data
  error = NULL;
  response =
    g_dbus_proxy_call_sync (proxy,
                            "org.freedesktop.DBus.Introspectable.Introspect",
                            NULL,
                            G_DBUS_CALL_FLAGS_NONE,
                            -1,
                            NULL,
                            &error);
  if (response == NULL)
    {
      return NULL;
    } // if (response == NULL)

  // Get the XML from the introspection data
  g_variant_get (response, "(&s)", &xml);
  // Build an object that lets us explore the introspection data.
  error = NULL;
  info = g_dbus_node_info_new_for_xml (xml, &error);
  g_variant_unref (response);
  if (info == NULL)
    {
      return NULL;
    } // if (info == NULL)

  // And return that object
  return info;
} // g_dbus_proxy_get_node_info
```
