

Developers manual: Implementing digital broadcasting enabled applications using the Openmokast framework

Jean-Michel Bouffard, M.A.Sc.
<jean-michel.bouffard@crc.ca>

December 2009

Developers manual version 0.1 (18-12-2009)
Applies to Openmokast software version 0.6.3

Contents

1	Preface	1
1.1	Introduction	1
1.2	Openmokast features	1
1.2.1	Inputs	1
1.2.2	Outputs	3
2	Development	5
2.1	Using the Openmokast graphical user interface	5
2.2	Using the openmokast Telnet API	5
2.3	Using the Openmokast DBus API.	10
3	Conclusions	14
	Bibliography	15

List of Figures

1.1	Openmokast receiver selection screen	2
2.1	Openmokast main window	6
2.2	Openmokast decoder selection window	7
2.3	Openmokast settings dialog	8
2.4	Openmokast API listed in D-Feet D-Bus debugger	11
2.5	Openmokast-client sample application	12

List of Tables

2.1 Available options when launching Openmokast	6
---	---

Nomenclature

API Application Programming Interface

GUI Graphical User Interface

MOT Multimedia Object Transfer

MSC Main Service Channel

RTP Real Time Protocol

Chapter 1

Preface

1.1 Introduction

The Openmokast application is a digital broadcasting player that is available to decode different types of DAB/DMB services. The player is a standalone application with a graphical user interface that enables users to tune to a frequency, discover the available services and decode some of the services.

One of the goal of the project was to enable new applications to be built over digital broadcasting channels and to foster innovation in that field. This manual explains how to take advantages of the Openmokast framework to build these new applications. Some control APIs (Application Programming Interfaces) have been integrated in the software and they can be used to easily take advantage of the framework to control a DAB/DMB receiver and to get access to the data streams of the different services.

The different software packages and the open source code of the application are available form the Openmokast website[3].

1.2 Openmokast features

1.2.1 Inputs

The software is compatible with different inputs that are loaded as dynamic modules at the application launch. The complete set of inputs are all available from the command line interface but some of the inputs are not included in the GUI (Graphical User Interface). The input selection screen is depicted in Fig. 1.1. The Openmokast input can be selected at the start of the application using the command line option mentioned in the following list. The compatible receivers are:

DCSR DSP device

- Module name: libdcsrdspinput
- Availability in Openmokast GUI: NA
- Usage in Command line: -D <device> DSP device name
- Description:

DCSR PCI input card

- Module name: libdcsrpciinput
- Availability in Openmokast GUI: NA
- Usage in Command line: -P <device> PCI device name
- Description:

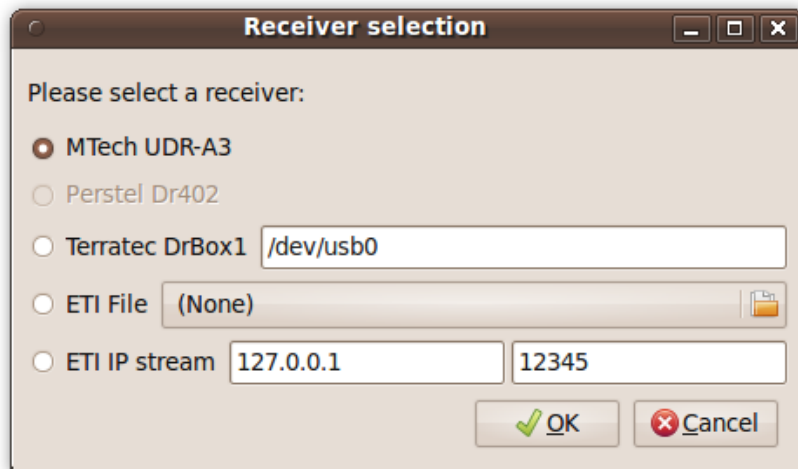


Figure 1.1: Openmokast receiver selection screen

DCSR USB device

- Module name: libdcusrinput
- Availability in Openmokast GUI: YES
- Usage in Command line: -U <device> USB device name
- Description: This device includes the Terratec DRBox1 DAB receiver.

MTech USB receiver

- Module name: libdabmateinput
- Availability in Openmokast GUI: YES
- Usage in Command line: -M dabMate device
- Description: This includes the MTech UDR-A3L DMB-T USB Digital Receiver and the Albrecht DR 403 - DAB/DMB Receiver.

Simulated input - File

- Module name: libetifileinput
- Availability in Openmokast GUI: YES
- Usage in Command line: -e <filename> Use ETI file as input
- Description: This input simulates a physical receiver from a file based data stream. Compatible streams can be generated online at [1].

Simulated input - TCP/IP

- Module name: libetitcpinput
- Availability in Openmokast GUI: YES
- Usage in Command line: -E <address:port> Use ETI data from a TCP connection as input
- Description: This input simulates a physical receiver from a TCP/IP based data stream. Compatible streams can be generated and transmitted using the CRC-DABMUX software from [1].

1.2.2 Outputs

The Openmokast outputs are also available as dynamic modules that are loaded at runtime of the application. The different outputs will either forward the stream over the network or decode the service directly. Stream forwarding can be used to send the content of a broadcast stream to another decoding application. Different outputs can be selected for each available subchannels. This selection can be achieved through the Telnet interface as it will be explained in section 2.2. The available outputs are:

File recorder

- Module name: libfile
- Availability in Openmokast GUI: YES
- DAB service compatibility: any
- Description: This will record the raw content of a subchannel to the specified file.

UDP/IP

- Module name: libudp
- Availability in Openmokast GUI: YES
- DAB service compatibility: any
- Description: This will retransmit the raw content of a subchannel over a UDP/IP socket to the specified destination.

RTP (Real Time Protocol)

- Module name: librtp
- Availability in Openmokast GUI: NA
- DAB service compatibility: Mpeg-2 audio
- Description: This will insert the Mpeg-2 audio stream into RTP packets and transmit the content to the specified destination.

HTTP

- Module name: libhttp
- Availability in Openmokast GUI: YES (the audio player in Openmokast is actually creating an HTTP socket that is opened by the external audio player)
- DAB service compatibility: any
- Description: This will create a simple HTTP server that will listen on the specified port. A request to the HTTP server will start streaming the raw content of the subchannel.

Celt audio decoder

- Module name: libceltplay
- Availability in Openmokast GUI: YES
- DAB service compatibility: CRC Celt audio service
- Description: This will extract Celt packets from the subchannel, decompress the audio and send it to the sound card through the Alsa audio subsystem if available.

MOT (Multimedia Object Transfer) decoder

- Module name: libmot
- Availability in Openmokast GUI: YES
- DAB service compatibility: any MOT application
- Description: This will extract the files inserted in the MOT service from a DAB packet data subchannel and write the file structure to a temporary folder (usually /tmp/MOT). The GUI application will show the extracted files as soon as they appear in the folder.

Journaline decoder

- Module name: libjournaline
- Availability in Openmokast GUI: YES
- DAB service compatibility: MOT Journaline
- Description: This will extract the text information inserted in the MOT-Journaline service from a DAB packet data subchannel and write text files to a temporary folder (usually /tmp/JOURNALINE). The GUI application has an internal browser that will present the content as it becomes available in the folder.

IP tunneling decoder

- Module name: libipoverdab
- Availability in Openmokast GUI: NA
- DAB service compatibility: DAB-IP Tunneling
- Description: This will try to send the IP packets contained in the IP Tunneling subchannel over a raw network socket. It may fail to do so if the destination IP address of the packets does not exist on the receiver side network.

Chapter 2

Development

2.1 Using the Openmokast graphical user interface

It is possible to have access to the broadcast bitstream only by using the Openmokast main application. This does not require any programming. The resulting data stream can be used to extract the data and to run any conversion or decoding algorithm. The two main mechanisms to get the stream are the file recorder and the UDP/IP forwarding.

In both cases, the data that will be extracted is the content of a raw subchannel. For stream audio and stream data type of subchannels, the raw content consists of the exact unmodified content of the subchannel. For a packet data type of subchannel, the recorded stream consists in a collection of packets that are received one after the other. In this case, to get the data of the actual service, the packets must be extracted and recombined into MSC Datagroups as in is explained in the DAB specification [4].

File recorder Recording of a broadcast subchannel in a file is a simple process. After the frequency is tuned, select the service component to record and click on the “Record” button which is shown in the Openmokast main GUI window of Fig. 2.1. After pressing the button, a file dialog will open to select the destination and the file name to record the stream to. The stream will be inserted into the file until the user clicks back on the record toggle button.

UDP/IP forwarding The UDP/IP forwarding will retransmit a broadcast subchannel over the network without any conversion nor modification. The destination of the UDP stream can be selected in the settings dialog of the application as shown in Fig. 2.3. Next, to start the streaming, select the service component and click on the “Play” button. The “Play” button brings up a dialog, shown in Fig. 2.2, that offers different playout options. The “UDP/IP stream” option will start the transmission.

This UDP/IP output stream can then be used easily by any third party decoder to process the content of the broadcast subchannel. The payload of the UDP packets represents the exact content of the subchannel without any synchronization or header bytes. The main advantage of the UDP/IP output is that it can be used to process the broadcast data in real time as it is received by the broadcast hardware.

2.2 Using the openmokast Telnet API

The Telnet API is more complex to use but it enables to do more things with Openmokast. One of the advantages of this interface is that it can be operated remotely. This enables the platform to run on a remote server and to be controlled from any other location. The Telnet interface can be used in collaboration with any of the other interfaces, but it can also be used alone. This means that Openmokast can be launched and controlled with the GUI and the Telnet interface at the same time but it can also be launched in command line only and controlled by Telnet alone. Two main usage models could be considered when using the software:

normal Usage by a single user who interacts with the software through the GUI.

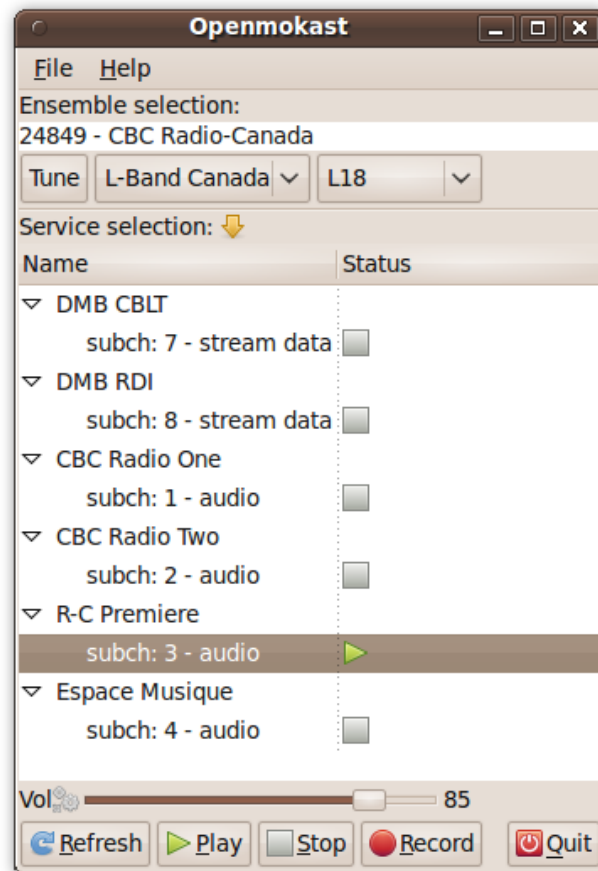


Figure 2.1: Openmokast main window

Command line option(s)	Result
> openmokast	Regular openmokast execution with GTK+ interface enabled.
> openmokast -c	Telnet mode only, no graphical interface. This mode can be used on a headless server.
> openmokast -c -p <telnet port>	Same as the previous but the Telnet server is listening to a port specified after the -p option (otherwise the default is 14000).

Table 2.1: Available options when launching Openmokast

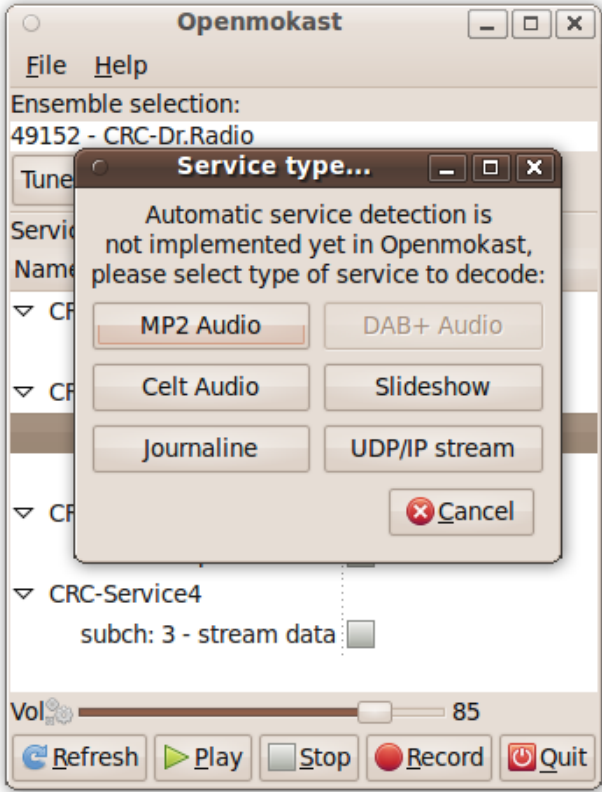


Figure 2.2: Openmokast decoder selection window

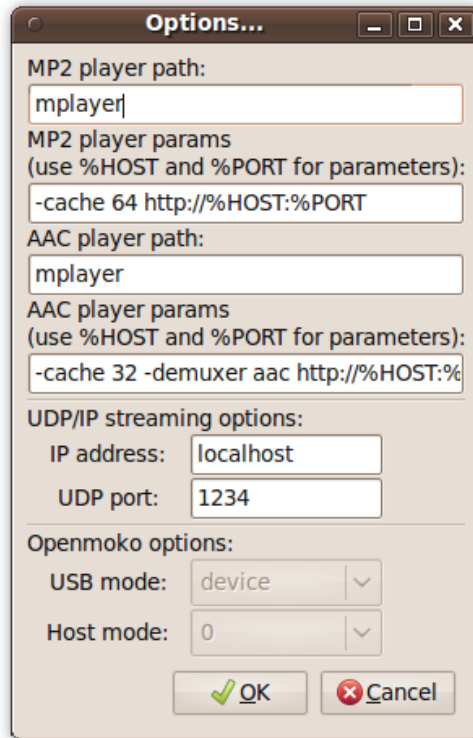


Figure 2.3: Openmokast settings dialog

server The software runs on an embedded device or a remote server and is controlled over the network.

The different ways to launched Openmokast are described in Table 2.1. After the Openmokast application is started using either execution model and options, the Telnet connection can be established using the “telnet” application which is common on many types of computer systems. In fact, Telnet is available by default on most GNU/Linux distributions and is also available on Windows™ 9x/2000/XP/Vista/7. If Openmokast running on the same machine as the telnet client and is listening the the default port, the following call will connect to the application:

```
user@user-pc:~/ $ telnet localhost 14000
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^['.
```

After the connection is established, it is possible to type the commands to control the receiver. The available commands can be listed by typing “help” but they are described in the following list:

List of available telnet command

```
tune [frequency] [mode]
    Tune the given frequency.
scan [band] [starting freq] : 1=Band-III, 2=L-Band, 3=Canada-L-Band(default), 4=
    Korea-Band-III
    Tune the first available frequency on the choosen band beginning at the
    starting freq.
scanensembles [band] : 1=Band-III, 2=L-Band, 3=Canada-L-Band(default), 4=Korea-
    Band-III
    Return a list of available ensembles.
getfrequency
```

```

        Return the current tuned frequency.
getdabstatus
        Get the current status of the DAB reception
getrdi
        Get all active rdi channels.
addsubchannel subch
        Enable data transfer for the specified subchannel.
removesubchannel subch
        Disable data transfer for the specified subchannel.
clearsubchannels
        Disable data transfer for all subchannels.
setdestination EId SId ServCompId destination port type
        Set the stream type and destination
        destination can be a IP address , a filename(-file-) or a network
        interface(-ip-)
        the type can be - rtp, http, udp, tcp, bridge, file, ip, dmb, mot -
setbaseaddress address
        Set the default address.
getbaseaddress
        Get the current default address.
getservices
        Return a list of available services.
getservcomp
        Get service components and output address and port.
getsubchannels
        Return a list of available subchannels.
getensembles
        Get the information about the tuned ensemble.
recordfic
        Enables FIC recording.
stopfic
        Disables FIC recording.
getficdestination
        Get the information about FIC output address and port.
setficdestination destination port type
        Set the FIC destination
        destination can be a IP address , a filename(-file-) or a network
        interface(-ip-)
        the type can be - rtp, http, udp, tcp, file, ip, journaline -
help
        This help.
kill
        Remotely kill CRC-DABRMS.

```

For instance, to control the software to stream a raw subchannel from the CBC in Canada over a UDP/IP socket, the following procedure must be executed. The CBC ensemble is broadcasting over the L18 frequency or 1482,464 MHz. First of all the Openmokast application must be started, next the Telnet interface will be used to connect to Openmokast and start the streaming with these commands:

```
> telnet localhost 14000
```

```
The following commands must be used under the telnet prompt (in the
current example, the CBC Radio One service will be streamed to udp://localhost
:1234):
```

```
% tune 1482464
```

```
% getservices
```

```
Ports Information :
```

```
EId SId Service Name
```

```

-----
24849 49168 CBC Radio One (RadioOne)
24849 49169 CBC Radio Two (RadioTwo)
24849 49170 R-C Premiere (Premiere)
24849 49171 Espace Musique (Espace M)
24849 2713714706DMB CBLT ()
24849 2713714707DMB RDI ()

% getservcomp
Services Components Information :
EId SId ServCompId SubChId TMid Component Name Protocole Destination
-----
24849 49168 1 1 0 CRC-SC 49168-1 udp localhost:1234
24849 49169 2 2 0 CRC-SC 49169-2 udp 127.0.0.1:13392
24849 49170 3 3 0 CRC-SC 49170-3 udp 127.0.0.1:13394
24849 49171 4 4 0 CRC-SC 49171-4 udp 127.0.0.1:13396
24849 27137147067 7 1 CRC-SC -15812525 dabplushttp 127.0.0.1:12390
24849 27137147078 8 1 CRC-SC -15812525 dabplushttp 127.0.0.1:12392

% setdestination 24849 49168 1 localhost 1234 udp
1-OK

% addsubchannel 1
1-OK

% exit
Connection closed by foreign host.

```

Next, any application that can listen to the UDP/IP stream could read the content of the subchannel and execute any required processing on the data.

2.3 Using the Openmokast DBus API

The DBus inter-process communication mechanism is becoming the standard for inter-process communication inside GNU/Linux systems. It enables any Linux application to offer its services to other applications on the same computer. The DBus API could be used to develop new broadcasting-enabled application or even to add support for broadcasting in existing applications.

The easiest way to discover the available DBus services over the system is to install the D-Feet DBus debugger software (For KDE based systems, the `kdbus` DBus service browser is also available). This application will connect on the bus and report any available services. Fig. 2.4 shows the Openmokast API that can be used over the bus. The specifications of the API and the documentation are available as a user friendly HTML version on the Openmokast website [3], in the “Development” section.

To use this API, an application only needs to link with the `libdbus` library as it is explained on `Freedesktop.org` [2]. Optionally, it is recommended to use high level bindings specific to particular languages and frameworks such as Qt, GLib, Java, C#, Python, and so on. Some samples of client code are available for a multitude of languages from the DBus website.

For development using GLib, a sample application that demonstrates how to use DBus to call some methods of the openmokast API was release along with Openmokast, including the complete source. The application is shown in Fig. 2.5. The header file `org.openmokast.Receiver.client.h`, needs to be included in any software projects that want to use Openmokast over DBus.

The client application must connect to the bus, and create a proxy object prior to be able to call DBus methods:

```

// Global variables
// DBus connection object
DBusGConnection *connection = NULL;
// DBus proxy object used to call Openmokast methods

```

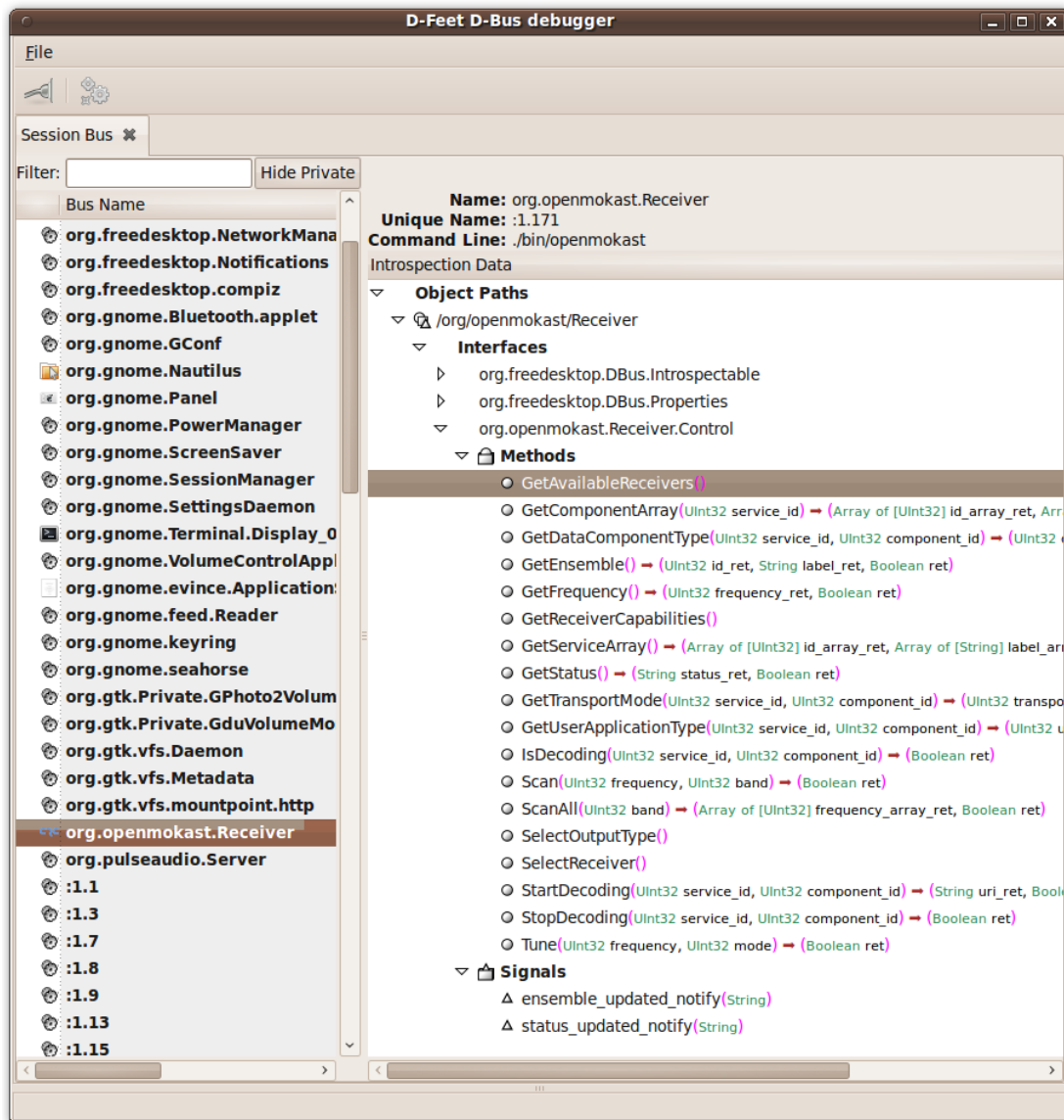


Figure 2.4: Openmokast API listed in D-Feet D-Bus debugger

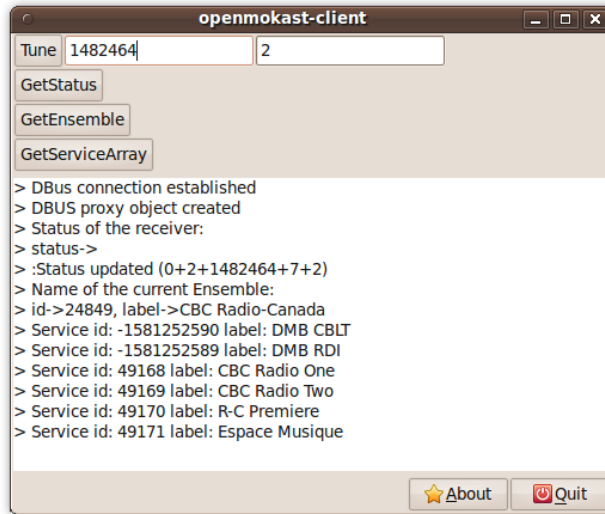


Figure 2.5: Openmokast-client sample application

```

DBusGProxy *proxy = NULL;

// DBus initialisation
GError *error;
gboolean test_ret;

g_type_init ();

error = NULL;
connection = dbus_g_bus_get (DBUS_BUS_SESSION, &error);
if (connection == NULL)
{
    g_printerr ("Failed to open connection to bus: %s\n",
                error->message);
    g_error_free (error);
}

/* Create a proxy object for the "openmokast server" (name "org.openmokast.
Receiver") */
proxy = dbus_g_proxy_new_for_name (connection,
    DBUS_SERVICE_OPENMOKAST,
    DBUS_PATH_OPENMOKAST,
    DBUS_INTERFACE_OPENMOKAST);
if (!proxy) {
    g_printerr ("Failed to create proxy...\n");
}

```

Next, it can start calling Openmokast methods over the bus through the proxy object, like shown here with the `GetServiceArray` method. Other Openmokast methods work similarly:

```

// Reads list of available services in ensemble
GError *error = NULL;
GArray *array1;
char **array2;
gboolean test_ret;
if (!org_openmokast_Receiver_Control_get_service_array (proxy,
    &array1,
    &array2,

```

```
&test_ret,  
&error))  
{  
    /* Checks for remote exceptions */  
    if (error->domain == DBUS_GERROR && error->code ==  
        DBUS_GERROR_REMOTE_EXCEPTION) {  
        g_printerr ("Caught remote method exception %s: %s",  
            dbus_g_error_get_name (error),  
            error->message);  
    }  
    else {  
        g_printerr ("Error: %s\n", error->message);  
    }  
    g_error_free (error);  
}
```

Chapter 3

Conclusions

These different APIs offer a flexible mechanism to control and use the Openmokast platform through other applications. This makes it easier to build broadcasting-enabled application. The Openmokast framework is also usable in an embedded environment as it was demonstrated with the FreeRunner version of the application.

Moreover, the different APIs can all be used at the same time. A user could then use the GUI to start a recording even if a video decoding software is using the DBus API to decode and play a DMB video service.

If you have question about development with Openmokast you are invited to ask it on our forum located at [1] in the section “mmbTools forum” -> “Openmokast” -> “Openmokast software stack”.

Bibliography

- [1] Crc mmb tools - an effort to support, foster and promote the development of new applications and tools in the field of mobile multimedia broadcasting (mmb). <http://mmbtools.crc.ca/>.
- [2] D-bus project page on freedesktop.org. <http://www.freedesktop.org/wiki/Software/dbus>. Last visited March 2009.
- [3] Openmokast - resources for open mobile broadcast devices. <http://openmokast.org/>.
- [4] Radio broadcasting systems; digital audio broadcasting (dab) to mobile, portable and fixed receivers. ETSI EN 300 401, 06 2006. V1.4.1.