

rtp-streaming with xine

Holger Schmuhl,
Tampere University of Technology

20.04.2003

Contents

1	Introduction	2
2	Linux Multimedia Players	2
2.1	Overview	2
2.2	MPlayer [3]	3
2.3	xine [5]	3
3	xine - what and how to change	4
3.1	changes concerning rtp-header	4
3.2	reading channel list	5
3.3	parsing channel list	8
3.3.1	format of channel list	8
3.3.2	channel structure	9
3.3.3	methode <code>play_item_load_ch ()</code>	10
3.3.4	methode <code>play_item_convert()</code>	13
3.4	changing the layout of the playlist	14
3.4.1	adding columns	14
3.4.2	adding channel items	16
4	Comments	17

1 Introduction

This report will first give an overview of the most popular Multimedia players for the Linux OS. It will then give a brief description of the players MPlayer and xine and its support of receiving rtp-streams. Finally it will focus on how the source code of xine has to be change to support the Digital TV Broadcast System of the Digital Media Institute (DMI) of the Tampere University of Technology (TUT).

2 Linux Multimedia Players

2.1 Overview

A few years ago, the missing multimedia capabilities of Linux were one of the main reasons which prevented users to change from MS Windows to the open source society. Only quite few audio and video formats where supported and also the output was not working properly or as wanted.

This changed due to a high development activity in this field. At the moment the most popular player in the audio field is **xmms** [1] and **xine** and **MPlayer** in the field of video. They support a great variety of audio and video formats and that is why they come with nearly every Linux distribution. To mention all the supported formats/standards e.g. of xine it will need at least one page and with nearly every new release, a few new formats/standards will join the list (that is why all the formats/standards are not mentioned in this document. To get detailed and up-to-date information, please refer to the corresponding website). The only plugin one might miss is the coffee-maker or tea-cooker plugin. At the moment there is no compareable player for MS Windows and if there will be one, it will be most proberly a port of the Linux players.

Concerning the support of video streaming in the field of Digital TV / Digital Video Broadcast over the network, the Ecole Central Paris has developed a quite well-engineered system called **VideoLAN** [2] which is supposed to run on most platforms.

The DMI at the TUT uses rtp-streams to broadcast Digital Satellite TV over the network. The task was to find and adjust a player which is able to receive rtp-streams under Linux. MPlayer, xine and also videoLAN were considered and xine was finally chosen, because it worked without nearly any problems and the required adjustments also seemed to be easy to implement in it.

2.2 MPlayer [3]

The latest version of MPlayer is 0.90 “CounterCounter” which was released at the beginning of April 2003. It is written in C and supposed to run on GNU/Linux, BSD, Sun Solaris, Silicon Graphics, Irix and QNX. Yet there exists no port to MS Windows.

MPlayer also supports rtp-streaming using the ”**LIVE.COM Streaming Media**” [4] source code libraries. A detailed description how to compile and configure MPlayer together with LIVE.COM can be found here: <http://www.live.com/mplayer/> (visited 04/2003).

I tested MPlayer version 0.90 rc 4 “FlameCounter” with the standard Mandrake 9.0 and SUSE 8.1 distributions. The compilation and execution of MPlayer and also the integration of the LIVE.COM Streaming Media libraries worked without any problem. But I was unable to receive and view rtp-streams because additional libraries were required to display them correctly. Due to this fact there occurred also problems to get along with more advanced video files, e.g. mpeg4 or divx files.

As MPlayer does not have any advantages across from xine and xine was working from the beginning on nearly perfectly, the choice was made in favour of xine.

2.3 xine [5]

The latest version of xine is **xine-lib-1-beta10** which was released at the 08.04.2003. The general architecture of xine can be characterized by a modular, advanced multi-threaded architecture. Beside the various plugins for input, demux, decode and output there are also several frontends available which make use of the xine-lib. Those frontends are written using the different GUI libraries of Linux. The most fully developed frontends seem to be **gxine** [5] (gtk+2.0, current version 0.3.3) and **xine-ui** [5] (Xlib, current version 0.9.20). Xine is written in C and supposed to run on GNU/Linux, BSD, Sun Solaris, Silicon Graphics and Irix. There is also a MS Windows version available [6], but it is outdated (last news 04/2002) and uses **xine-lib 0.9.8** which has no working support for receiving rtp-streams. In general it should be possible to get gxine together with the latest version of xine-lib working under MS Windows, because there is also a **gtk+** runtime and development environment available for MS Windows [7].

The reception of rtp-streams works smooth since **xine-lib-1-beta7**.

The following paragraph will focus on how to adjust xine to work properly with rtp-streams in the environment of the DMI at the TUT.

3 xine - what and how to change

I performed and tested the changes with the standard Mandrake 9.0 and SUSE 8.1 distributions on ix86 systems. All the mentioned changes and code sections refer to xine-lib-1-beta10 and gxine 0.3.3.

Because I was working with several different versions of xine, I installed them all in my home directory and not in the standard system directory. This can be done by executing the following commands, which add the specified path to the corresponding shell variables so that gxine will find xine-lib in the right directory:

```
export PATH="$HOME/<xine directory>/bin:$PATH"
export LD_LIBRARY_PATH="$HOME/<xine directory>/lib:$LD_LIBRARY_PATH"
export CFLAGS="-I$HOME/<xine directory>/include -L$HOME/<xine directory>/lib
-03"
export ACLOCAL_FLAGS="-I $HOME/<xine directory>/share/aclocal"
```

and by calling the configure command with the following parameter:

```
./configure --prefix=$HOME/<xine directory>
```

After that `make install` will automatically install the executives in the subdirectory `bin/` of `<xine directory>/`.

As I was using on one system a quite old ATI graphic card, some strange crashes of xine occurred from which it did not clearly result that the problem was caused by the XFree driver of the card. After I used the plain X11 output device `xshm` instead of the much faster `xv`, everything was working fine.

In the following paragraphs, the modifications which have to be done to get xine working with the rtp-streams used in environment of the DMI at the TUT are described in detail. Most of them were only necessary to read a list of the available channels from a server and to paste them in the playlist of the frontend gxine.

Hint: If one wants to understand how xine works in general, how the coherences are between plugins, demuxer, decoder etc. and where to add additional code, functions etc., the “Hacker’s Guide” which can also be found on the homepage [5] is very helpful.

3.1 changes concerning rtp-header

The rtp-header used at the DMI is not 100% conform to the header specified in the referring RFC 1889 [8]. This requires slight changes in the method `static void * input_plugin_read_loop(void *arg)` in the file `input_rtp.c` located in the directory `src/input/` of the xine-lib-1-beta10 as shown below:

```

static void * input_plugin_read_loop(void *arg) {

// ... non-relevant code removed

pad = data[0] & 0x20;
ext = data[0] & 0x10;
csrc = data[0] & 0x0f;

// ... non-relevant code removed

}

```

must be changed into:

```

static void * input_plugin_read_loop(void *arg) {

// ... non-relevant code removed

pad = 0;
ext = 0;
csrc = 0;

// ... non-relevant code removed

}

```

3.2 reading channel list

A list of all channels which are broadcasted via rtp is available from a server at the DMI. It can be accessed via a standard tcp socket connection.

Normally gxine only reads the media sources from a file called `playlist` located in the subdirectory `.gxine/` of the home-directory. The implementation of gxine was changed in that way, that if the file `playlist` contains only the keyword `tvlist` followed by the IP-address of the server where the actual playlist (containing all the available channels broadcasted via rtp) can be retrieved from, they are read from port 4444 of this server. A possible improvement of this methode would be to include also a DNS lookup procedure. The file `playlist` should than look like the following example:

```
tvlist 130.230.50.30
```

The implementation can be seen below. The changes were performed in the method `static int playlist_load (char *fname)` of the file `playlist.c` located in directory `src/` of `gxine`.

```
// ... non-relevant code removed

#include <sys/socket.h>
#include <netinet/in.h>

// ... non-relevant code removed

static int playlist_load (char *fname) {

    char *plfile;

    if (!fname)
        fname = g_strconcat(g_get_home_dir(), "/.gxine/playlist", NULL);

    plfile = read_entire_file_ascii (fname);

    // ... inserted code starts here

    // copy of the actual plfile pointer to
    // run through the content
    char *plfile2;

    //allocate space for pl2, the
    //length of plfile+ the extra '\0' character
    plfile2 = (char *) malloc (strlen (plfile) + 1);
    strcpy (plfile2, plfile);
    char *keyword = "tvlist";
    // at the moment, servername must still contain the
    // ip address of the server
    char *servername;
    char *temp = strtok (plfile2, " ");
    if (strcmp (temp, keyword) == 0)
    {
        servername = strtok (NULL, " ()\t\n\f\r\0");
        printf ("Servername: %s\n", servername);
        free (plfile);
        int sock, size, x;
```

```

plfile = (char *) malloc (65536);
struct sockaddr_in s_i;
sock = 0;
memset (&s_i, 0, sizeof (s_i));
sock = socket (AF_INET, SOCK_STREAM, 0);
s_i.sin_family = AF_INET;
inet_aton (servername, &(s_i.sin_addr));
s_i.sin_port = htons (4444);

// memset(&(s_i.sin_zero), '\0', 8);
if (connect(sock, (struct sockaddr *) &s_i,sizeof (s_i)) != 0)
{
printf ("unsuccessful connect! \n");
return -1;
}
printf ("sock=%d\n", sock);

if ((x =read (sock, &size, sizeof (size))) != sizeof (size))
{
printf ("x=%d\n", x);
puts ("error reading size");
return -1;
}
printf ("size = %d\n", size);
x = 0;
while ((x += read (sock, plfile + x, size - x)) < size) ;
close (sock);
// printf ("content of buffer:%s\n", buf);
}

free (plfile2);

// ... inserted code ends here

// ... non-relevant code removed

```

To assure that the playlist is not overwritten when xine is being quitted, the current playlist is saved with the name `playlist_last`. So it can be easily restored via the GUI. The changes were performed in the methode `void playlist_save (char *fname)` of

the file `playlist.c` located in directory `src/` of `gxine`:

```
// ... non-relevant code removed

void playlist_save (char *fname) {

    FILE *f;

    if (!fname)
        // the file is saved with a different name, so that
        // .gxine/playlist still contains the keyword and address of
        // the server which provides the channel-list
        fname = g_strconcat(g_get_home_dir(), ".gxine/playlist_last",
        NULL);

    f = fopen (fname, "w");

    // ... non-relevant code removed

}
```

3.3 parsing channel list

3.3.1 format of channel list

The standard playlist of `gxine` is in xml-format. The channel list which is read from the server is also in xml-format, but it contains different information. Only the following format of the channel list can be read, otherwise it will lead to an error message:

```
<xml>
  <channel>
    <name>Al Jazeera</name>
    <vpid></vpid>
    <apid></apid>
    <country></country>
    <language>Arabic</language>
    <type></type>
    <ip>239.252.4.10</ip>
    <port>5555</port>
    <FECport>5556</FECport>
    <tptype>mcast</tptype>
```

```

        <svtype>rtp</svtype>
    </channel>
</xml>

```

3.3.2 channel structure

To handle the channel information, a separate structure called `play_item_c` is defined, which is similar to the normal `play_item_s` structure. The only difference is that it contains more attributes. It is defined as shown below in the file `play_item.h` located in directory `src/` of `gxine`:

```

// ... non-relevant code removed

typedef struct play_item_s play_item_t;
typedef struct play_item_c play_item_ch;

struct play_item_s {

    char    *title;
    char    *mrl;

    int     start_time;

    GList   *options; /* options are simply script
                       engine command strings */

    int     played; /* bool indicating this has
                    been played in playlist */
};

struct play_item_c {

    char    *title;
    char    *language;
    char    *type;
    char    *mrl; // should be assembled by:
                 // svtype + "://" + ip + ":" + port

    char    *ip;
    char    *port;
    char    *FECport;
    char    *svtype;

```

```

    int    start_time;

    GList *options; /* options are simply script
                    engine command strings */

    int    played; /* bool indicating this has
                    been played in playlist */
};

// ... non-relevant code removed

```

3.3.3 methode `play_item_load_ch ()`

This methode was added to the file `play_item.c` located in directory `src/` of `gxine` to read the channel information from the parse tree (which will be explained in 3.4.2, page 16) in the channel structure. It also converts the given data into a valid MRL¹.

Below is a table which shows how the data fields of the xml channel list are patched in the `play_item.c` structure and later on in the columns of the `gxine` playlist:

xml	play_item.c	referring column in the playlist of gxine
name	title	title
language	language	language
type	type	description
ip	ip	MRL
port	port	MRL
FECport	FECport	-
svtype	svtype	MRL

It returns an object of the type `play_item.c`. The implementation is given below:

```

// ... non-relevant code removed

play_item_ch *play_item_load_ch (xml_node_t *node) {

```

¹MRL: media resource locator

```

play_item_ch *play_item;

play_item = malloc (sizeof (play_item_ch));

play_item->title      = NULL;
play_item->language   = NULL;
play_item->ip         = NULL;
play_item->port       = NULL;
play_item->FECport    = NULL;
play_item->svtype     = NULL;
play_item->options    = NULL;
play_item->mrl        = NULL;
play_item->start_time = 0;

while (node) {

    if (!strcasecmp (node->name, "name")) {
        play_item->title = strdup (node->data);
#ifdef LOG
        printf ("play_item: title = %s\n",
            play_item->title);
#endif
    } else if (!strcasecmp (node->name, "vpid")) {
        // do nothing
    } else if (!strcasecmp (node->name, "apid")) {
        // do nothing
    } else if (!strcasecmp (node->name, "country")) {
        // do nothing
    } else if (!strcasecmp (node->name, "language")) {
        play_item->language = strdup (node->data);
#ifdef LOG
        printf ("play_item: language = %s\n",
            play_item->language);
#endif
    } else if (!strcasecmp (node->name, "type")) {
        play_item->type = strdup (node->data);
#ifdef LOG
        printf ("play_item: type = %s\n", play_item->type);
#endif
    } else if (!strcasecmp (node->name, "ip")) {

```

```

        play_item->ip = strdup (node->data);
#ifdef LOG
        printf ("play_item: ip   = %s\n", play_item->ip);
#endif
    } else if (!strcasecmp (node->name, "port")) {
        play_item->port = strdup (node->data);
#ifdef LOG
        printf ("play_item: port   = %s\n", play_item->port);
#endif
    } else if (!strcasecmp (node->name, "FECport")) {
        play_item->FECport = strdup (node->data);
#ifdef LOG
        printf ("play_item: FECport   = %s\n",
                play_item->FECport);
#endif
    } else if (!strcasecmp (node->name, "tptype")) {
        // do nothing
    } else if (!strcasecmp (node->name, "svtype")) {
        play_item->svtype = strdup (node->data);
#ifdef LOG
        printf ("play_item: svtype   = %s\n",
                play_item->svtype);
#endif
    }
    // needed?
    else if (!strcasecmp (node->name, "time")) {
        play_item->start_time =
xml_parser_get_property_int (node, "start", 0);
#ifdef LOG
        printf ("play_item: start = %d\n", play_item->start_time);
#endif
    } else
        printf ("play_item: error while loading, unknown node %s\n",
                node->name);

    node = node->next;
}

// build mrl
play_item->mrl =
(char*)calloc(1, strlen(play_item->svtype) +

```

```

        strlen(play_item->ip) +
        strlen(play_item->port) + 5);
strcpy(play_item->mrl, play_item->svtype);
strcat(play_item->mrl, "://");
strcat(play_item->mrl, play_item->ip);
strcat(play_item->mrl, ":");
strcat(play_item->mrl, play_item->port);

return play_item;
}

// ... non-relevant code removed

```

3.3.4 methode `play_item_convert()`

This methode is required, because gxine reads the actual information of the media source from a normal `play_item`, which is inserted in an invisible column of the playlist. It returns a `play_item_s` structure (see 3.3.2, page 9), which is a conversion of `play_item_c`, given to it as a parameter. It is implemented in the file `play_item.c` located in directory `src/` of gxine. The refering source code is given below:

```

// ... non-relevant code removed

play_item_t *play_item_convert(play_item_ch *toConvert) {

    play_item_t *play_item;

    play_item = malloc (sizeof (play_item_t));

    play_item->title      = NULL;
    play_item->options    = NULL;
    play_item->mrl        = NULL;
    play_item->start_time = 0;

    // convert the channel item to a normal one
    play_item->title = (char*)calloc(1, strlen(toConvert->title)
        + strlen(toConvert->language) +
        4);
    strcpy(play_item->title, toConvert->title);
}

```

```

    strcat(play_item->title, " (");
    strcat(play_item->title, toConvert->language);
    strcat(play_item->title, ")");
    play_item->mrl = (char*)calloc(1, strlen(toConvert->mrl) +
        1);
    strcpy(play_item->mrl, toConvert->mrl);

    return play_item;
}

// ... non-relevant code removed

```

3.4 changing the layout of the playlist

The standard playlist which is available in gxine offers only columns for the title and the referring MRL of the media source. For the user it would be also helpful to know in which language the channel is broadcasted and of which type it is. So it was decided to add additional columns to the playlist containing those informations.

3.4.1 adding columns

The GUI and so the columns of the playlist are created by the methode `void playlist_init (void)` in the file `playlist.c`.

To keep the implementation compatible, so that also non-channel items can still be inserted properly in the playlist, the numeration of the existing columns was left as it was. The additional columns where numbered successively, the referring source code is given below:

```

// ... non-relevant code removed

void playlist_init (void) {

// ... non-relevant code removed

    /*
     * init tree store
     */

    /*

```

```

    * to assure the compatibility with the rest of the code, this
order and
    * and numeration of the columns has been chosen instead of:
    * pl_store = gtk_list_store_new (3, G_TYPE_STRING,
    * G_TYPE_STRING, G_TYPE_POINTER);
    */
pl_store = gtk_list_store_new (5, G_TYPE_STRING, G_TYPE_STRING,
    G_TYPE_POINTER, G_TYPE_STRING, G_TYPE_STRING);

/*
    * tree view widget to display playlist
    */

tree_view = gtk_tree_view_new_with_model
(GTK_TREE_MODEL(pl_store));
gtk_tree_view_set_rules_hint (GTK_TREE_VIEW (tree_view), TRUE);
g_signal_connect (G_OBJECT(tree_view), "button_press_event",
    G_CALLBACK(button_press_lcb), NULL);

cell = gtk_cell_renderer_text_new ();
column = gtk_tree_view_column_new_with_attributes ("title",
    cell,
    "text", 0,
    NULL);
gtk_tree_view_append_column (GTK_TREE_VIEW (tree_view),
    GTK_TREE_VIEW_COLUMN (column));

// added
column = gtk_tree_view_column_new_with_attributes ("language",
    cell,
    "text", 3,
    NULL);
gtk_tree_view_append_column (GTK_TREE_VIEW (tree_view),
    GTK_TREE_VIEW_COLUMN (column));

//added
column = gtk_tree_view_column_new_with_attributes ("description",
    cell,
    "text", 4,
    NULL);
gtk_tree_view_append_column (GTK_TREE_VIEW (tree_view),

```

```

        GTK_TREE_VIEW_COLUMN (column));

column = gtk_tree_view_column_new_with_attributes ("mrl",
        cell,
        "text", 1,
        NULL);
gtk_tree_view_append_column (GTK_TREE_VIEW (tree_view),
        GTK_TREE_VIEW_COLUMN (column));

gtk_tree_view_set_reorderable (GTK_TREE_VIEW (tree_view), TRUE);

// ... non-relevant code removed

}

```

3.4.2 adding channel items

The parse tree mentioned before is created by the call of the functions `xml_parser_init` (`plfile`, `strlen (plfile)`, `XML_PARSER_CASE_INSENSITIVE`) and `xml_parser_build_tree` (`&node`) in the method `static int playlist_load (char *fname)` of the file `playlist.c`. Each channel is loaded from it with the help of the method `play_item_ch *play_item_load_ch (xml_node_t *node)` (see 3.3.3, page 10). The result is then converted to a normal `play_item` (see 3.3.4, page 13). After that the channel item is added to the GUI. The referring section of the source code is given below:

```

static int playlist_load (char *fname) {

// ... non-relevant code removed

if (plfile) {

    xml_node_t *node;

    xml_parser_init (plfile, strlen (plfile),
XML_PARSER_CASE_INSENSITIVE);

    if (xml_parser_build_tree (&node)>=0) {

        if (!strcasecmp (node->name, "asx")) {

```

```

// ... non-relevant code removed

}

}
// in case it is the channel list
else if (!strcasecmp (node->name, "xml"))
{
playlist_clear ();
node = node->child;
while (node)
{
if (!strcasecmp (node->name, "channel"))
{
play_item_ch * play_item_chan = play_item_load_ch(node->child);
play_item_t * play_item = play_item_convert(play_item_chan);
GtkTreeIter iter;
gtk_list_store_append(pl_store, &iter);
gtk_list_store_set (pl_store, &iter,
0, play_item_chan->title,
1, play_item->mrl,
2, play_item,
3, play_item_chan->language,
4, play_item_chan->type, -1);
}
node = node->next;
}
}

else {

// ... non-relevant code removed

```

4 Comments

I performed this project as an Erasmus student at the TUT during the spring semester of 2003. It comes without any warranty. If you have any questions feel free to contact me: holger.schmuhl@epost.de .

Thanks to Irek Defee, Florin Lohan, Marius Vlad and Aurelian Pop for the assistance

and the opportunity to perform this project and to the xine development team for this great multimedia player!

References

- [1] X MultiMedia System
<http://www.xmms.org> (visited 04/2003)
- [2] VideoLAN, Ecole Central Paris
<http://www.videolan.org> (visited 04/2003)
- [3] MPlayer
<http://www.mplayerhq.hu> (visited 04/2003)
- [4] LIVE.COM Streaming Media
<http://www.live.com> (visited 04/2003)
- [5] xine
<http://www.xinehq.de> (visited 04/2003)
- [6] xine for win32
<http://www.matthewgrooms.net/> (visited 04/2003)
- [7] GTK+ for Windows
<http://www.dropline.net/gtk/> (visited 04/2003)
- [8] RTP - RFC1889
<http://www.faqs.org/rfcs/rfc1889.html> (visited 04/2003)