Subject: Sound in linux
Posted by rainbowsally on Tue, 16 Dec 2014 09:51:19 GMT
View Forum Message <> Reply to Message

These are freedesktop sounds but I don't know what installed them. They were in my suse 11.4 and my mint 15

/usr/share/sounds/freedesktop/stereo/dialog-error.oga
/usr/share/sounds/freedesktop/stereo/dialog-warning.oga
/usr/share/sounds/freedesktop/stereo/dialog-information.oga

Probably the most generic player with ability to suppress commandline noise is sox. It installs the /usr/bin/play program.

Here's what I have for uppsrc/Core/Util.cpp for the linux sounds.

```
#ifdef PLATFORM_POSIX
static void LinuxBeep(const char *fn)
{
// return;
// // This is not the right way to do that... (causes zombies,
// ignores Gnome settings)
 char hb[100];
 char* h = hb; // so we can see the string in the debugger
 sprintf(h, "play -q /usr/share/sounds/freedesktop/stereo/dialog-%s.oga 2>/dev/null &", fn);
// strcat(h, fn);
//#ifdef CPU_BLACKFIN
// if(vfork()) return;
//#else
// if(fork()) return;
//#endif
// IGNORE_RESULT(
  0 == system(h);
// );
// _exit(EXIT_SUCCESS);
}
#endif

void BeepInformation()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONINFORMATION);
#else
 LinuxBeep("information");
#endif
}
```

```
void BeepExclamation()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONEXCLAMATION);
#else
 LinuxBeep("warning");
#endif
}

void BeepQuestion()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONQUESTION);
#else
 LinuxBeep("information");
// write(1, "\a", 1); //??
#endif
}
```

UPP is so cool it hurts. :) Thank you.

Subject: Re: Sound in linux
Posted by mirek on Wed, 17 Dec 2014 10:30:16 GMT
View Forum Message <> Reply to Message

rainbowsally wrote on Tue, 16 December 2014 10:51These are freedesktop sounds but I don't know what installed them. They were in my suse 11.4 and my mint 15

/usr/share/sounds/freedesktop/stereo/dialog-error.oga
/usr/share/sounds/freedesktop/stereo/dialog-warning.oga
/usr/share/sounds/freedesktop/stereo/dialog-information.oga

Probably the most generic player with ability to suppress commandline noise is sox. It installs the /usr/bin/play program.

Thanks, I have taken your code and started investigating; ended with something like this:

#ifdef PLATFORM_POSIX

```
String CurrentSoundTheme = "freedesktop";

static void LinuxBeep(const char *name)
{
 String fn = "/usr/share/sounds/" + CurrentSoundTheme + "/stereo/dialog-" + name;
 system("play -q  " + fn + (FileExists(fn + ".ogg") ? ".ogg" :
                            FileExists(fn + ".oga") ? ".oga" :
                            FileExists(fn + ".wav") ? ".wav" :
                            ".*")
        + " >/dev/null 2>/dev/null&");
}

#endif

void BeepInformation()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONINFORMATION);
#else
 LinuxBeep("information");
#endif
}

void BeepExclamation()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONEXCLAMATION);
#else
 LinuxBeep("warning");
#endif
}

void BeepError()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONERROR);
#else
 LinuxBeep("error");
#endif
}

void BeepQuestion()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONQUESTION);
#else
 LinuxBeep("question");
#endif
```

}

then in ChGtk:

```
CurrentSoundTheme = GtkStyleString("gtk-sound-theme-name");
```

seems to work fine, playing sounds from selected theme. So from now on, thanks to you, U++ finally started to beep in Linux :)

Mirek

Subject: Re: Sound in linux
Posted by rainbowsally on Wed, 17 Dec 2014 17:54:09 GMT
View Forum Message <> Reply to Message

The changes in Core/Util.cpp work, but I got errors when I dropped the other snippet into ChGtk.cpp. It looks like a snippet from inside a function?

So I commented it out for now.

Works. :)

Finding the right place to plug it in may be a little confusing for others so in case anyone else wants to play with this, I'll upload the uppsrc/Core/Util.cpp per your changes here.

File Attachments

1) Util.cpp, downloaded 319 times

Subject: Re: Sound in linux
Posted by mirek on Wed, 17 Dec 2014 18:56:15 GMT
View Forum Message <> Reply to Message

rainbowsally wrote on Wed, 17 December 2014 18:54The changes in Core/Util.cpp work, but I got errors when I dropped the other snippet into ChGtk.cpp. It looks like a snippet from inside a function?

So I commented it out for now.

Works. :)

Finding the right place to plug it in may be a little confusing for others so in case anyone else wants to play with this, I'll upload the uppsrc/Core/Util.cpp per your changes here.

No worry, it is already in trunk and next nightly build... :)

Subject: Re: Sound in linux (& generic launchers)
Posted by rainbowsally on Thu, 18 Dec 2014 03:23:57 GMT
View Forum Message <> Reply to Message

Yuh know?

The lack of standards in Linux is really a problem.

Here's the problem, linux guys.

ls /usr/share/sounds/freedesktop/stereo/*-warning.*
ls /usr/share/sounds/freedesktop/stereo/*-information.*
ls /usr/share/sounds/freedesktop/stereo/*-question.*

See it? The question sound (a symlink to one of the basic sounds) is inconsistent with the other two dialog sounds and is probably misnamed.

This is the sound string I'd recommend for "question" that should have the greatest probability of being portable.

```
void BeepQuestion()
{
#ifdef PLATFORM_WIN32
 MessageBeep(MB_ICONQUESTION);
#else
 LinuxBeep("warning");
#endif
}
```

There may be a ton of interesting sounds for KDE and possibly GTK in the directory above these, but as far as standards go, freedesktop.org is the closest we've got to a standard at this point.

////////// And Generic Launchers ////////

[There's a question at the bottom.]

There are MANY more problems with linux GUI toolkits, though. And I'm not sure that this is the place to bring them up, but in GUIs we expect things to run concurrently. We don't expect things like playing sounds to block (that is, to wait until the sound is finished) before painting a window. And we don't want to have to create a new library of new non-standard sounds that blow our applications up in size due to the impracticality of compressing these things.

There are ways to accomplish this by way of shell calls, but (again) there is no basic binutils or coreutils level of linux utility to do this.

Examples of applications that do shell calls that run concurrently, by way of reparenting the called process to 'init' are kshell3, 4, or 5. I have kshell4. It genuinely returns before the sound is finished playing.

In kde compare [Note the lowercase Q switch. Don't use P].
play -q /usr/share/sounds/KDE-Sys-Log-In-Long.ogg
to
kshell4 play -q /usr/share/sounds/KDE-Sys-Log-In-Long.ogg
and if you try the launch utility, try that as well.
launch play -q /usr/share/sounds/KDE-Sys-Log-In-Long.ogg

And for superuser, there kdesu, kdesudo, gksu, and xdg-su, and others which also reparent the called application to init. But they are meant for running GUIs from the commandline as superuser. That's overkill and in fact not at all what we want.

Here's a simple app, that returns immediatly, generically, leaving a very small footprint in memory while the called shell runs.

I confess, I haven't run valgrind on it and that might not be the only problem with it, though I have used it for years and will continue to use it unless or until something better turns up.

I called it "launch". And you probably won't like it. But I love it. :)

Here's the main function.

```
int main(int argc, char** argv)
{
  int err = 0;

  // help and version switches
  if(argc == 1)
    return usage(0);

  if(argc == 2)
  {
    if(strcmp(argv[1], "--help") == 0)
```

```
    return usage(0);
  if(strcmp(argv[1], "-v") == 0)
    return print_version(0);
}

err = parse_options(argc, argv);
if(err)
{
  printf("Unknown launch option '%s'\n", argv[err]);
}

// reassemble commandline
char args[1024];
err = unparse_args(args, 1000, argc, argv);
if(err)
{
  if(err == ENOMEM)
    fprintf(stderr, "Out of memory or args list too long\n");
  else if (err == E2BIG)
    fprintf(stderr, "Argument list too long\n");
  return 1;
}

// if -q suppress all feedback from the app
if(!flg.warn)
{
  fclose(stdout);
  fclose(stderr);
  strcat(args, " > /dev/null 2>&1");
}

// do the fork and execute the commandline
pid_t pid;

pid = fork ();
if (pid == 0)
{


  // command path, command name, arg1, arg2, ...
  execl ("/bin/sh", "sh", "-c", args, NULL);

  // _exit(code) for threads and forks.
  _exit (EXIT_FAILURE);
}
else if (pid < 0)
  return -1;
else
```

```
    return 0;
}
```

And I'll attach the missing pieces as a tarball in case anyone is curious about this thing or has any ideas as to how to improve it.

The binary and sources, and makefile are all included. I believe the executable is the debug version.

If you watch the processes with gnome-system-monitor or ksysguard (or ps -ax or ???) you will see that the application runs under init and the footprint is very reasonable. The switches allow feedback but the default is to run silently.

Question: Is it possible to launch an invisible window to run these "shells" in UPP so that we get concurrently running applications without having to use external tools such as kshell<N> or 'launch" while retaining the ability to shut them down and possibly kill applications that hang if they exceed a reasonable timeout?

File Attachments

1) launch-1.4.tar.gz, downloaded 236 times

Subject: Re: Sound in linux (& generic launchers)
Posted by dolik.rce on Thu, 18 Dec 2014 05:35:45 GMT
View Forum Message <> Reply to Message

rainbowsally wrote on Thu, 18 December 2014 04:23Question: Is it possible to launch an invisible window to run these "shells" in UPP so that we get concurrently running applications without having to use external tools such as kshell<N> or 'launch" while retaining the ability to shut them down and possibly kill applications that hang if they exceed a reasonable timeout?
Hi,

There is no need for an invisible window. Just remember the pid returned from fork and call kill on it after the timeout is exceeded. In GUI apps, you can do this easily using SetTimeCallback().

Best regards,
Honza

Subject: Re: Sound in linux
Posted by rainbowsally on Fri, 19 Dec 2014 13:45:18 GMT
View Forum Message <> Reply to Message

Hi Honza.

Quote:
There is no need for an invisible window. Just remember the pid returned from fork and call kill on it after the timeout is exceeded. In GUI apps, you can do this easily using SetTimeCallback().

Technically, you're right, but getting GUIs and the old terminal based linuxes is not so straight forward.

For example how could you get the PID without "waitpid()" or reading a return string from a system() call.

The calling app would hang so you'd never get a chance to shut it down.

A system call to an intermediate shell utility similar to kshell<N> is a much better alternative for a simple dialog bell of some sort.

Either it runs or it doesn't.

But I see now that the "invisible window" thing wouldn't work either. Not sure how this "shell" utility could be integrated into UPP. Maybe run the fork in another thread. Then I suppose that thread's binary image would be all that gets copied to memory.

It's going to take me a while to get up to speed here.

Thanks for UPP and this forum. :)

-rs

Subject: Re: Sound in linux
Posted by dolik.rce on Sun, 21 Dec 2014 20:04:50 GMT
View Forum Message <> Reply to Message

rainbowsally wrote on Fri, 19 December 2014 14:45Hi Honza.

Quote:
There is no need for an invisible window. Just remember the pid returned from fork and call kill on it after the timeout is exceeded. In GUI apps, you can do this easily using SetTimeCallback().

Technically, you're right, but getting GUIs and the old terminal based linuxes is not so straight forward.
In U++ there is even ready to use solution for non-GUI apps. See bazaar/Timer. Or you could write your own, based on thread, or POSIX alarms, or any from many other suitable technologies.

rainbowsally wrote on Fri, 19 December 2014 14:45For example how could you get the PID without "waitpid()" or reading a return string from a system() call.

The calling app would hang so you'd never get a chance to shut it down.
The fork() call in your previous example returns the pid of child process to the parent. There is no need to block on the waitpid, it should only be called to collect the finished processes. If you don't want it to block your application, you can use it with the NOHANG option.

rainbowsally wrote on Fri, 19 December 2014 14:45A system call to an intermediate shell utility similar to kshell<N> is a much better alternative for a simple dialog bell of some sort.

Either it runs or it doesn't.
Calling shell just to run a command is in most cases unnecessary overkill. The shell actually does pretty much the same thing as described above.

Honza

Subject: Re: Sound in linux
Posted by rainbowsally on Mon, 22 Dec 2014 07:01:59 GMT
View Forum Message <> Reply to Message

Quote:

Calling shell just to run a command is in most cases unnecessary overkill. The shell actually does pretty much the same thing as described above.

Honza

Possibly.

But let's say you launch a shell with a command to.. let's say 'find' everything on your C drive and dump it into a text file.

Anything, just something that takes a long time would do.

If this shell is launched by way of a stystem() call from a GUI, will the GUI hang until 'find' is done?

Seems to me it does. Maybe not with the '&' at the end though.

In QT the buttons didn't pop back up until the system call was finished.

[I'm not being lazy and not checking this out, I'm just too busy at the moment.]

Subject: Re: Sound in linux
Posted by dolik.rce on Mon, 22 Dec 2014 08:56:26 GMT
View Forum Message <> Reply to Message

rainbowsally wrote on Mon, 22 December 2014 08:01But let's say you launch a shell with a command to.. let's say 'find' everything on your C drive and dump it into a text file.

Anything, just something that takes a long time would do.

If this shell is launched by way of a stystem() call from a GUI, will the GUI hang until 'find' is done?

From the man page of system():
Quote:

> The system() library function uses fork(2) to create a child process that executes the shell command specified in command using execl(3) as follows:
>
> execl("/bin/sh", "sh", "-c", command, (char *) 0);
>
> system() returns after the command has been completed.

So to answer your question, yes it blocks. But you can always call fork() and one of the exec* functions (there is about six of them). That in itself is not blocking, later you can check if the process is completed by calling waitpid(pid_returned_from_fork, &return_code, NOHANG), which is also nonblocking operation.

Honza

Subject: Re: Sound in linux
Posted by rainbowsally on Tue, 23 Dec 2014 08:40:29 GMT
View Forum Message <> Reply to Message

Hi Honza.

Quote:From the man page of system():

> The system() library function uses fork(2) to create a child process that executes the shell command specified in command using execl(3) as follows:
>
> execl("/bin/sh", "sh", "-c", command, (char *) 0);
>
> system() returns after the command has been completed.

So to answer your question, yes it blocks. But you can always call fork() and one of the exec* functions (there is about six of them). That in itself is not blocking, later you can check if the

process is completed by calling waitpid(pid_returned_from_fork, &return_code, NOHANG), which is also nonblocking operation.

Honza

True. :)

Now here's a trickier one. How might we launch an app in another thread, that we KNOW will take a long time, and shut it down with calls to STOP and CONTINUE from time to time, in order to assure that the GUI continues to be responsive during a system() call.

Subject: Re: Sound in linux
Posted by rainbowsally on Tue, 23 Dec 2014 12:32:08 GMT
View Forum Message <> Reply to Message

Honza, one more thing.

Threads are disabled in Linux by default. I think I know why, and I think there's an easy fix (re. the intermittent crash problem and thread timing), but at this point, we can't count on running in a separate thread working for all non-windows users.

I have enabled threads for myself though and eventually will get around to experimenting with them.

I'll check out the UPP AltThreads package in a few minutes.

Subject: Re: Sound in linux
Posted by Klugier on Tue, 23 Dec 2014 13:49:28 GMT
View Forum Message <> Reply to Message

Hello,

I think there is a problem with this code. "play" command is not install by default on my distribution (Kubuntu), so the sound cannot be played. Alternatively, I can use command "ogg123" to play .ogg audio file.

So, I am almost sure that above solution is not distribution independent and should be improved for example if "play" is not detected use "ogg123" instead.

Sincerely,
Klugier

Subject: Re: Sound in linux
Posted by rainbowsally on Wed, 24 Dec 2014 03:40:00 GMT
View Forum Message <> Reply to Message

Hi Klugier.

On my system, I don't have ogg123 (at this time). I have gst123 (a gstreamer version with a bit more flexibility).

There's so many variations in linux. And xdg-open won't play anything if there's no mime-link set for it.

Your solution works, but it's probably even less generic than 'play'.

"play" is in the sox package and handles all kinds of different sound formats, not just ogg/vorbis stuff. It's a commandline utility but it is very handy to convert between audio formats.

But if you've got what you need now, you probably don't need to get sox. It's a bit overkill, though it is quite useful at times.

Klugier wrote on Tue, 23 December 2014 14:49Hello,

I think there is a problem with this code. "play" command is not install by default on my distribution (Kubuntu), so the sound cannot be played. Alternatively, I can use command "ogg123" to play .ogg audio file.

So, I am almost sure that above solution is not distribution independent and should be improved for example if "play" is not detected use "ogg123" instead.

Sincerely,
Klugier

Subject: Re: Sound in linux
Posted by mirek on Wed, 24 Dec 2014 11:27:32 GMT
View Forum Message <> Reply to Message

Klugier wrote on Tue, 23 December 2014 14:49Hello,

I think there is a problem with this code. "play" command is not install by default on my distribution (Kubuntu), so the sound cannot be played. Alternatively, I can use command "ogg123" to play .ogg audio file.

So, I am almost sure that above solution is not distribution independent and should be improved for example if "play" is not detected use "ogg123" instead.

Sincerely,
Klugier

You are welcome to propose the patch. Anyway, at the moment, using 'play' means going from being silent on all distros to being silent on just some. I guess that is an improvement :)

Mirek

Subject: Re: Sound in linux
Posted by Klugier on Wed, 24 Dec 2014 12:18:51 GMT
View Forum Message <> Reply to Message

Hello Mirek,

The detection can be done in following way:

```
// Copy from PrinterJob.cpp -> I think it should be part of POSIX U++ library!!!
static String System(const char *cmd, const String& in)
{
 String ofn = GetTempFileName();
 String ifn = GetTempFileName();
 SaveFile(ifn, in);
 String c = cmd;
 c << " >" << ofn;
 if(in.GetCount())
  c << " <" << ifn;
 String q;
 if(system(c) >= 0)
  q = LoadFile(ofn);
 FileDelete(ofn);
 FileDelete(ifn);
 return q;
}

static String FindPlayer()
{
 static String player;

 if (player.IsEmpty()) {
  const char *players[] = { "play", "ogg123" }; // <- Add all possible players here!!!

  for (int i = 0; i < __countof(players); i++) {
   if (!System("which " + String(players[i]), Null).IsEmpty()) {
    player = players[i];
    break;
```

```
  }
 }
}

 return player;
}

static void LinuxBeep(const char *name)
{
 String player = FindPlayer();
 if (!player.IsEmpty()) {
  String fn = "/usr/share/sounds/" + CurrentSoundTheme + "/stereo/dialog-" + name;
  system(player + " -q " + fn + (FileExists(fn + ".ogg") ? ".ogg" :
                                 FileExists(fn + ".oga") ? ".oga" :
                                 FileExists(fn + ".wav") ? ".wav" :
                                 ".*")
         + " >/dev/null 2>/dev/null&");
 }
}
```

IMO, There is a problem that function "System" is not part of U++ (I don't want to do copy&past). Can we do something with that issue?

Sincerely,
Klugier

File Attachments

1) Sound.diff, downloaded 229 times

Subject: Re: Sound in linux
Posted by mirek on Wed, 24 Dec 2014 14:46:45 GMT
View Forum Message <> Reply to Message

Maybe use Sys instead?

Subject: Re: Sound in linux
Posted by Klugier on Wed, 24 Dec 2014 15:30:26 GMT
View Forum Message <> Reply to Message

Hello Mirek,

It seems that "Sys" works perfect...

```
static String FindPlayer()
{
 static String player;
 if (player.IsEmpty()) {
  const char *players[] = { "play", "ogg123", "gst123" };
  for (int i = 0; i < __countof(players); i++) {
   String out;
   if (Sys("which " + String(players[i]), out) == 0 && !out.IsEmpty()) {
    player = players[i];
    break;
   }
  }
 }
 return player;
}

static void LinuxBeep(const char *name)
{
 String player = FindPlayer();
 if (!player.IsEmpty()) {
  String fn = "/usr/share/sounds/" + CurrentSoundTheme + "/stereo/dialog-" + name;
  system(player + " -q " + fn + (FileExists(fn + ".ogg") ? ".ogg" :
                                 FileExists(fn + ".oga") ? ".oga" :
                                 FileExists(fn + ".wav") ? ".wav" :
                                 ".*")
         + " >/dev/null 2>/dev/null&");
 }
}
```

I enclose also diff...

But, all this applications needs user installation. So, sound out of the box (without installation any package) seems to be hard. We can also make additional dependency for example with "sox" package (Debian base distribution).

Next problem is GCC warning, because we ignore system return value. What should we do?

```
int status = system(...)
if (status == -1) return; // <- Throw exception???
```

Sincerely,
Klugier

File Attachments
1) Sound.diff, downloaded 242 times

Subject: Re: Sound in linux
Posted by mirek on Thu, 25 Dec 2014 09:48:33 GMT
View Forum Message <> Reply to Message

Thanks, modified version of this code applied to trunk...

Mirek

Subject: Re: Sound in linux
Posted by Klugier on Thu, 25 Dec 2014 12:56:32 GMT
View Forum Message <> Reply to Message

Hello Mirek,

I think you should also protect the situation when player string is empty. It can leads to meaningless system call, that cost more resources than normal function call. The system needs to detect that invoking command is wrong.

Code:

```
if (!player.IsEmpty()) {
  String fn = "/usr/share/sounds/" + CurrentSoundTheme + "/stereo/dialog-" + name;
  // Call only if player is detected!!!
              IGNORE_RESULT(system(player + " -q " + fn +
              (FileExists(fn + ".ogg") ? ".ogg" :
               FileExists(fn + ".oga") ? ".oga" :
                  FileExists(fn + ".wav") ? ".wav" :
                    ".*")
             + " >/dev/null 2>/dev/null&"));
}
```

Sincerely,
Klugier

Subject: Re: Sound in linux
Posted by rainbowsally on Thu, 25 Dec 2014 12:57:23 GMT
View Forum Message <> Reply to Message

Hi Klugier and Mirek and Linux developers.

Klugier wrote on Wed, 24 December 2014 16:30Hello Mirek,

Next problem is GCC warning, because we ignore system return value. What should we do?

```
int status = system(...)
if (status == -1) return; // <- Throw exception???
```

Sincerely,
Klugier

Klugier: No throwing an exception wouldn't be helpful. Just ignore the result. Warnings should be shown at compile time, if at all.

Mirek: also, note that the "question" for the question sound in the freedesktop files is window-question.* not dialog-question.* so the code posted here a while back, as written might not find the right file.

I may start checking this stuff out more dillegently in the future... Still figuring out how stuff works.

All: We don't want to hard-code something like ogg123 as the handler or even as a possible handler for the sound files if we plan to ever handle any other extension than .oga or .ogg because ogg123 only handles ogg.

We don't even want it to be an option because if we play something with some other extension it won't work at best and it might hang (zombie) or cause a horrible clicking white-noise racket by playing the wrong format.

This "racket" does happen with other players trying to play mp3s, not sure about ogg123.

Mirek: this brings us back to the serious problem of lack of standards in linux. "play" from the sox package handles practically every format except mp3. Or perhaps we need to develop our own sounds and the player for them. But then, of course, we have the fun task of detecting the system driver, of which there are currently about 5 non-standards to try to detect.

Sox (including which includes "/usr/bin/play") is indeed overkill, but it is very complete and already handles all this linux non-standards nonsense.

PS. Sorry if I misremember any of the names of the files or macros.

Subject: Re: Sound in linux
Posted by Klugier on Thu, 25 Dec 2014 13:23:04 GMT

Hello rainbowsally,

Quote:
All: We don't want to hard-code something like ogg123 as the handler or even as a possible handler for the sound files if we plan to ever handle any other extension than .oga or .ogg because ogg123 only handles ogg.

We don't even want it to be an option because if we play something with some other extension it won't work at best and it might hang (zombie) or cause a horrible clicking white-noise racket by playing the wrong format.

This "racket" does happen with other players trying to play mp3s, not sure about ogg123.

ogg123 can detect that it cannot play .wav file. It not hangs and not produce noise (Checked by playing wav file). So, we can change the importance of players (* gst123 can handle all interesting formats):

const char *players[] = { "play", "gst123", "ogg123" };

Moreover, I would like to notice that having more trusted players in list we have more likely that someone has got this program.

Sincerely,
Klugier

Subject: Re: Sound in linux
Posted by mirek on Thu, 25 Dec 2014 20:01:46 GMT

You are right.

Mirek
