Xdialog

Exercises

Prepare by
Ahmed Bentiba

For

ECE Linux User Group

May 6, 2003

Comment: Some examples of using Xdialog and we will discuss them later on

```
#!/bin/bash
#Example 1

Xdialog --title "Example 1: " \
--backtitle " Date:`date`" \
--msgbox "Welcome to Xdialog!" 15 50 8


#!/bin/bash
#Example 2

Xdialog --title "Example 2 " \
--backtitle " Date:`date`" \
--infobox "The Xdialog is nice and easy to use \n The parametres \
 -1 -1 on the right mean the outpout will cover the whole monitor \
 you can also write it as -1x-1 \n This infobox will close automatically." -1 -1


#!/bin/bash
#Example 3

Xdialog --title "Example 3 " \
--backtitle " Date:`date`" \
--infobox "Output of the widget somewhere on the screen" 0x0+120-45
#0 means auto-sized widget
#+120 means 120 pixel from the Left edge of the screen
#-45 means 45 pixel from the Bottom edge of the screen

#!/bin/bash
#Example 4

Xdialog --title "Example 4: Yesno " \
--backtitle " Date:`date`" \
--yesno "Do you like this tool?" 0x0
#0 means auto-sized widget


#!/bin/bash
#Example 5
Xdialog --title "Example 5: Input Box " \
--backtitle " Date:`date`" \
--inputbox "What is your name?" 0x0 2>/tmp/name.$$
#Xdialog --textbox  /tmp/name.$$  0 0
Xdialog --msgbox  `cat /tmp/name.$$`  0 0


#!/bin/bash
#Example 6


Xdialog --title "Example 6: 2 Input Box " \
--backtitle " Date:`date`" \
--password --2inputsbox "Welcome to the ECE Linux Users Group" 0x0 "Login" "ahmed"
"password" "ece"  2>/tmp/name.$$
Xdialog --textbox  /tmp/name.$$  0 0
#Xdialog --msgbox  `cat /tmp/name.$$`  0 0
```

```
#!/bin/bash
#Example 7

Xdialog --title "Example 7: Edit a file " \
--backtitle " Edit a File" \
--editbox example0 0 0
#no saving allowed, just to read the file


#!/bin/bash
#Example 8
touch /tmp/file.$$
FILE="/tmp/file.$$"
echo -e "\\033[1;31mThis is a red foreground text." >> $FILE
echo "Welcome to Xdialog!" >> $FILE
echo -e "\\033[1;38mRevert to normal text." >>$FILE
echo "Welcome to Xdialog!" >> $FILE
echo -e "\\033[1;31mANother color." >> $FILE
echo -e "\\033[1;36mWhy bother with colors?." >> $FILE
echo -e "\\033[1;42mThis is a green background text." >>$FILE
echo "Welcome to Xdialog!" >> $FILE
echo -e "\\033[1;35mAnother line!" >>FILE
echo -e "\\033[1;48mRevert to normal text." >> $FILE
echo "Normal text." >> $FILE
echo "Welcome to Xdialog!" >> $FILE

Xdialog --title "Example 8: Colors " \
--backtitle "Make colors in your text files!" \
--logbox $FILE 0 0


#!/bin/bash
#Example 9

Xdialog --title "Example 9: Checklist and Radio Buttons " \
--backtitle "Lot of choices" \
--checklist "What are your favorite fruits?" 0 0 5 \
"1" "Date" "on" \
"2" "Orange" "off" \
"3"  "Apple"  "off" \
"4"  "Watermelon" "on"
Xdialog --radiolist "What is your favorite fruit?" 0 0 5 \
"1" "Date" "on" \
"2" "Orange" "off" \
"3"  "Apple"  "off" \
"4"  "Watermelon" "off"


#!/bin/bash
#Example 10
# main() {

Xdialog --title "Example 1: Linux Tool" \
--backtitle "User:`whoami`, Date:`date`" \
--menu "Do not throw your money through Windows; use Linux!" 0 0 0 \
"1" "Memory status"  \
"2" "File System Status"  \
"3"  "Users Logged Now on this Machine"  \
"4" "Users Logged in the past on this Machine"  \
"Q" "Quit"  2> /tmp/ex1.$$
```

```
if [ $? = 1 -o $? = 255 ] ; then
    /bin/rm /tmp/ex1.$$
    exit
fi

choice=`cat  /tmp/ex1.$$`
 /bin/rm -f  /tmp/ex1.$$


if [ "$choice" = "1" ] ; then
   free > /tmp/free.tmp.$$

 Xdialog --title  "Memory status" \
--textbox  /tmp/free.tmp.$$  18 70

/bin/rm -f  /tmp/free.tmp.$$

fi

if [ "$choice" = "2" ] ; then
   df > /tmp/df.tmp.$$

  Xdialog --title  "File System Status" \
--textbox  /tmp/df.tmp.$$  18 76

/bin/rm -f  /tmp/df.tmp.$$

fi
if [ "$choice" = "3" ] ; then
   users > /tmp/users.tmp.$$

  Xdialog --title  "Users logged now" \
--textbox  /tmp/users.tmp.$$  18 76

/bin/rm -f  /tmp/users.tmp.$$

fi
if [ "$choice" = "4" ] ; then
   last > /tmp/last.$$

 Xdialog --title  "Users Logged in this Machine" \
--textbox  /tmp/last.tmp.$$  18 76
/bin/rm -f  /tmp/last.tmp.$$
fi

if [ "$choice" = "Q" ] ; then
   clear
   exit
fi



#!/bin/bash
#Example 11
 main() {
Xdialog --title "Example 1: Linux Tool" \
--backtitle "User:`whoami`, Date:`date`" \
--menu "Do not throw your money through Windows; use Linux!" 0 0 0 \
"1" "Memory status"  \
"2" "File System Status"  \
```

```
"3"  "Users Logged Now on this Machine"  \
"4" "Users Logged in the past on this Machine"  \
"Q" "Quit"  2> /tmp/ex1.$$

if [ $? = 1 -o $? = 255 ] ; then
    /bin/rm /tmp/ex1.$$
    exit
fi

choice=`cat  /tmp/ex1.$$`
 /bin/rm -f  /tmp/ex1.$$


if [ "$choice" = "1" ] ; then
   free > /tmp/free.tmp.$$

 Xdialog --title  "Memory status" \
--backtitle "Watch out!" \
--textbox  /tmp/free.tmp.$$  18 70

/bin/rm -f  /tmp/free.tmp.$$

fi

if [ "$choice" = "2" ] ; then
   df > /tmp/df.tmp.$$

  Xdialog --title  "File System Status" \
--backtitle "Space is never enough!" \
--textbox  /tmp/df.tmp.$$  18 76

/bin/rm -f  /tmp/df.tmp.$$

fi
if [ "$choice" = "3" ] ; then
   users > /tmp/users.tmp.$$

  Xdialog --title  "Users logged now" \
--textbox  /tmp/users.tmp.$$  18 76

/bin/rm -f  /tmp/users.tmp.$$

fi
if [ "$choice" = "4" ] ; then
   last > /tmp/last.$$

 Xdialog --title  "Users Logged in this Machine" \
--backtitle "Be careful!" \
--textbox  /tmp/last.tmp.$$  18 76
/bin/rm -f  /tmp/last.tmp.$$
fi

if [ "$choice" = "Q" ] ; then
   clear
   exit
fi

main
}
main
```

```
#!/bin/bash
#Example 12
# Another way to loop a menu
while true
do

Xdialog --title "Example 1: Linux Tool" \
--backtitle "User:`whoami`, Date:`date`" \
--menu "Do not throw your money through Windows; use Linux!" 0 0 0 \
"1" "Memory status"  \
"2" "File System Status"  \
"3"  "Users Logged Now on this Machine"  \
"4" "Users Logged in the past on this Machine"  \
"Q" "Quit"  2> /tmp/ex1.$$

if [ $? = 1 -o $? = 255 ] ; then
    /bin/rm /tmp/ex1.$$
    exit
fi

choice=`cat  /tmp/ex1.$$`
 /bin/rm -f  /tmp/ex1.$$


if [ "$choice" = "1" ] ; then
   free > /tmp/free.tmp.$$

 Xdialog --title  "Memory status" \
--backtitle "Watch out!" \
--textbox  /tmp/free.tmp.$$  18 70

/bin/rm -f  /tmp/free.tmp.$$

fi

if [ "$choice" = "2" ] ; then
   df > /tmp/df.tmp.$$

  Xdialog --title  "File System Status" \
--backtitle "Space is never enough!" \
--textbox  /tmp/df.tmp.$$  18 76

/bin/rm -f  /tmp/df.tmp.$$

fi
if [ "$choice" = "3" ] ; then
   users > /tmp/users.tmp.$$

  Xdialog --title  "Users logged now" \
--textbox  /tmp/users.tmp.$$  18 76

/bin/rm -f  /tmp/users.tmp.$$

fi
if [ "$choice" = "4" ] ; then
   last > /tmp/last.$$

 Xdialog --title  "Users Logged in this Machine" \
--backtitle "Be careful!" \
--textbox  /tmp/last.tmp.$$  18 76
```

```
/bin/rm -f  /tmp/last.tmp.$$
fi

if [ "$choice" = "Q" ] ; then
   clear
   exit
fi

done


#!/bin/bash
# Example 13
# Another way to loop a menu
main()
{
while [ 0 ] ; do

Xdialog --title "Example 13: Using Functions" \
--backtitle "User:`whoami`, Date:`date`" \
--menu "Do not throw your money through Windows; use Linux!" 0 0 8 \
"1" "Memory status"  \
"2" "File System Status"  \
"Q" "Quit"  2> /tmp/ex1.$$

if [ $? = 1 -o $? = 255 ] ; then
    /bin/rm /tmp/ex1.$$
    exit
fi

choice=`cat  /tmp/ex1.$$`
 /bin/rm -f  /tmp/ex1.$$


if [ "$choice" = "1" ] ; then
  memory   # call the function memory(), do not put () after the function name
fi

if [ "$choice" = "2" ] ; then
  fileSystem   # call the function fileSystem()
fi

if [ "$choice" = "Q" ] ; then
   exit
fi

done
}

memory()
{
while [ 0 ] ; do

 free > /tmp/free.$$

 Xdialog --title  "Memory status" \
--backtitle "Watch out!" \
--textbox  /tmp/free.$$  18 70
/bin/rm -f  /tmp/free.$$
```

```
main  # back to main

done
}

fileSystem()
{
while [ 0 ] ; do
 df > /tmp/df.$$

  Xdialog --title  "File System Status" \
--backtitle "Space is never enough!" \
--textbox  /tmp/df.$$  0 0

/bin/rm -f  /tmp/df.$$

main   # back to main
done
}


users()
{
while [ 0 ] ; do

users > /tmp/users.$$

  Xdialog --title  "Users logged now" --textbox  /tmp/users.$$  18 76

/bin/rm -f  /tmp/users.$$

main  # back to main
done
}


while true
do
main
done
```

Xdialog documentation - Introduction

- **What is Xdialog?**

  Xdialog is designed to be a drop in replacement for the dialog and cdialog programs (BTW Xdialog may also be used in place of gdialog). It converts any terminal based program into a program with an X-windows interface. The dialogs are easier to see and use and the new widgets add even more functionnalities.

  To achieve this, Xdialog uses the Gimp ToolKit (also known as GTK+). This toolkit is now widely used and it is most probably already installed on your system.

- **Why should I use it?**

  There are a few reasons for using Xdialog:

  - It's a drop-in replacement for (c)dialog and as such, it will give you a better interface for the existing scripts using (c)dialog at the cost of very little effort (if any).
  - Many things can be done from the command line (shell) under UNIX, but nowadays this (typing commands in a shell and using shell scripts) is not considered as "user-friendly". Xdialog gives you an opportunity to revamp your useful shell scripts so that any user can easily make use of them (he will not even notice they are scripts, he will just see the pretty GUI).
  - You may already have considered writing a program (in C or any other high level language) to do some simple things (things that can be done using simple UNIX commands), but writing such a program with a proper GUI takes time and requires a good knowlegde in high level programming... therefore you just abandonned the idea. Now you can use any shell together with Xdialog and write your program/utility in a few minutes!

- **What are the limits of Xdialog usage?**

  Xdialog is only able to set up (relatively) simple dialogs. It is in no way designed to write a full application with a proper window, menu bar and sub-menus. Therefore any task requiring more than a few prompts from the user and/or displaying complex data will most likely needs for something more powerful than Xdialog (i.e. you will have to do some true programming in high level language, and use GTK+, Motif or Qt toolkits).

- **Could you spot some typical applications for Xdialog?**

  Here are some examples of what I am using Xdialog for (either at home or at work):

  - Software installation scripts (an installation script using Xdialog makes for a good "installation wizard").
  - Customized and more powerful replacements for a few Xwindows utility such as xless or xmessage.
  - Wrappers and GUI for command line only UNIX utilities (playmidi, xanim, qiv, ping, fwhois, nslookup, etc...).
  - PPP (multiple ISPs) connection wizard/PPP log displayer (that is some sort of customized gppp/kppp).
  - Utility scripts for shell-less user accounts.

- Configuration wizard for software without configuration tool/menu.
- User mounting utility, encrypted loop mounting utility.
- Easy, semi-automatic Linux kernel compilation/installation.

- **What are the system requirements IOT execute Xdialog?**

  They are very minimal (no exotic/silly library needed): X11 and GTK+ (v1.2.0 or upper, v1.2.8 and upper preferred) is all what Xdialog needs to run. Of course you will also need a shell (any shell: sh, bash, ksh, csh, zsh...) or a scripting language (that is a language able to call external commands, e.g.: Perl).

  If you are using the Linux x86 pre-compiled RPM packages available at http://xdialog.free.fr, then please note that they were compiled against GTK+ v1.2.10, glibc v2.1.3 and XFree86 v4.1.0 (they are dynamically linked).

Xdialog documentation - General syntax and usage

Usage:

Xdialog [<GTK+ options>] [<common options>] [<transient options>] <box option>...

And:

Xdialog <special option>

The [<common options>] [<transient options>] <box option> sequence may be repeated several times in the same Xdialog command line (this is called **dialog chaining**). The common/transient options may be omited but there must always be a <box option> as the last Xdialog option.

- The <common options> are options applying to all following <box options> until the same or opposite <common options> are encountered into the Xdialog command line. These options are mostly dealing with menu look, style, placement and behavior.
- The <transient options> only apply to the next <box option> into the Xdialog command line. These options are used to tune the widgets (number and type of buttons, menu icon) or to trigger some of the widgets specific features.
- The <box option> tells to Xdialog which widget must be used and is followed by three or more parameters:
  - the first parameter is a text string or a filename (this depends on the box option);
  - the second and the third parameters are menu <height> and <width> in characters;
  - Some box options require additional parameters such as tags, menu items, labels or default values.

  On completion of each box option (i.e. every time a widget is closed) Xdialog sends any result (text, numbers) as one or more strings to stderr (this can be changed so that the results are sent to stdout thanks to a common option).

When Xdialog terminates (i.e. when all the <box options> are processed or when an error occurs), the exit code may take the following values:

- **0** : *OK*, *Yes* or *Next* button pressed.
- **1** : *Cancel* or *No* button pressed.
- **2** : *Help* button pressed (see the --help transient option for details).
- **3** : *Previous* button pressed (see the --wizard transient option for details).
- **255** : an error occurred or the box was closed through the window manager (same exit code as when the ESC key is pressed in (c)dialog).

Note that when chaining dialogs, the chain is broken and Xdialog terminates as soon as a widget returns a non zero exit code.

When using Xdialog from a shell, it is therefore usually invoked as follow:

```
RESULTS=`Xdialog --stdout ...` # It is also possible to redirect Xdialog output to a temporary file.
EXIT_CODE=$?
case $EXIT_CODE in
  0) # All OK. The $RESULTS variable holds everything entered/choosed by the user.
    .../...
    ;;
  1) # Cancel/No pressed.
    .../...
    ;;
255) # An error occurred or the box was closed.
    .../...
    ;;
esac
```

For examples of how to use each widget, please read the box options section of this documentation and browse the samples directory.

- The <special options> take no parameter and are to be used alone into the command line. They just make Xdialog to print a string on stderr and to exit immediately (with a 0 exit code).

Xdialog documentation - Common options

The **common options** apply to all the following box options until the same or opposite **common options** are encountered into the Xdialog command line; each common option either got one opposite or several complementary options, or allow to reset the parameter they change to its default value; their effect may therefore later be cancelled or changed for the other box options that follow in a chained dialogs command line.

Available common options and parameters:

- --wmclass <name>
-
- --rc-file <gtkrc filename>
-

- --backtitle <backtitle>
- --title <title>
- --allow-close | --no-close
- --screen-center | --under-mouse | --auto-placement
- --center | --right | --left | --fill
- --no-wrap | --wrap
- --cr-wrap | --no-cr-wrap
- --stderr | --stdout
- --separator <character> | --separate-output
- --buttons-style default|icon|text

- **--wmclass** <name>

  This option allows to set the window manager class name for Xdialog so that its window may be differenciated and decorated differently from other applications windows. This, of course, will depend on your window manager features... This also allows to use different decorations for different scripts using Xdialog. The **--wmclass** option must be followed by a <name> (the wmclass name of Xdialog then becomes <name>/<name>).

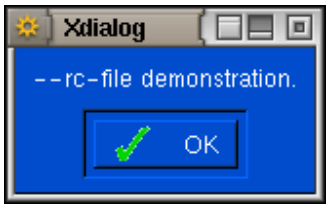
- **--rc-file** <gtkrc filename>

  This option allows to change the GTK+ theme for the following box options into the Xdialog command line. This option must be followed by the name of a file in the gtkrc format. As an example, the following lines may be put into a rc file which name will be passed to Xdialog after the **--rc-file** option; the result will be blue Xdialog boxes with white text:

  bluebox.rc file contents:

```
style 'blue_background' {
        bg[NORMAL] = { 0.0, 0.3, 0.8 }
        fg[NORMAL] = { 1.0, 1.0, 1.0 }
}
widget '*' style 'blue_background'
```

  Result:

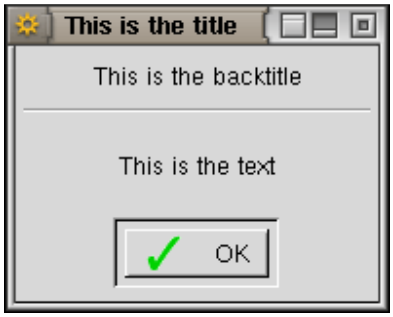


  Using the bluebox.rc file:

```
Xdialog --rc-file bluebox.rc --msgbox "--rc-file demonstration." 0 0
```

- **--backtitle** <backtitle>

  Title, backtitle and text positions in Xdialog windows:

  In (c)dialog, this option is used to print a title on the screen (terminal) background of the dialog boxes, thus the name... As it is not very friendly nor even desirable to print text on the X11 screen background (AKA "root window"), Xdialog uses this option to print a title **into** its widgets, at the top of them (horizontally centered). The default is no backtitle and you may restore this default after using **--backtitle** once into a chained dialog command line, by passing an empty string to the next **--backtitle** option as parameter. When the widget accepts a <text> parameter and when the backtitle is not an empty string, Xdialog automatically adds an horizontal line as a separator between the backtitle and the text. As for the <text> parameter, the <backtitle> parameter may contain "\n" (line feeds) characters. This option applies to all widgets.

- **--title** <title>

  This option sets the title of the Xdialog **window**. The title appears into the title bar which position depends on the window manager you are using. If no **--title** option is given then the title defaults to "Xdialog".

- **--allow-close** | **--no-close**

  --allow-close is the default and entitles the user to close the Xdialog window through the window manager; when a close event is received by Xdialog, it exits immediately and returns 255 as the exit code to the shell. Specifying **--no-close** will make Xdialog ignore any *close* (sometimes called *delete*) event originating from the window manager. Note that it is always possible, although not recommended, to *destroy* a Xdialog window using the window manager destroy event.

- **--screen-center** | **--under-mouse** | **--auto-placement**

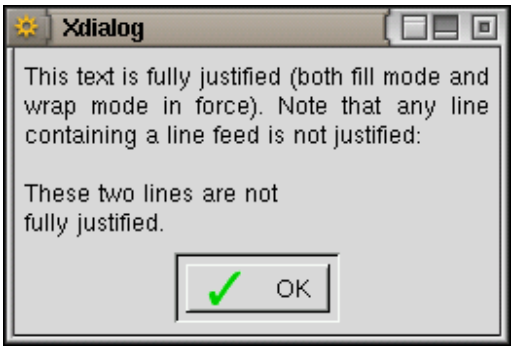
  These options control the Xdialog widgets placement (ditto). The default is **--screen-center**. The **--auto-placement** option lets the window manager decide where to pop up Xdialog widgets. Note that some window managers may perfectly ignore these options (which are only requests sent by GTK+ to the window manager) and place the widgets where they feel like...

- **--center** | **--left** | **--right** | **--fill**

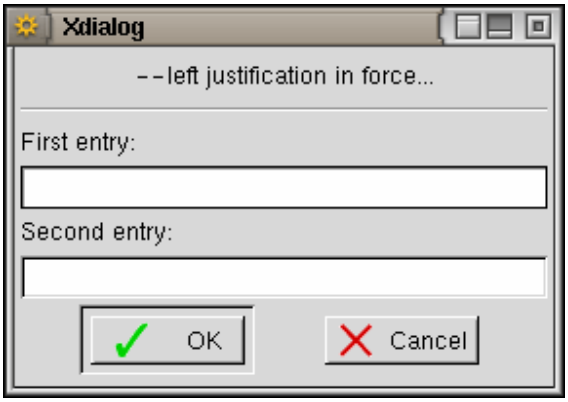
  These options instruct Xdialog to justify the text centered, left, right or full into the <text> label. Default is **--center**. The effect of these options is only sensible for multiple-lines text.

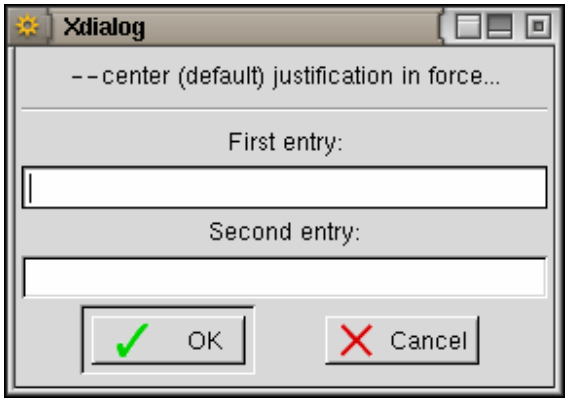
  Because of what I consider a bug in GTK+, the center and right justification modes only work when the GTK+ line wrap mode is not in force, while the fill justification only works when the GTK+ line wrap mode is in force (go figure...). This is why, as of v2.0.0, Xdialog uses its own wrapping routine for all cases but when the **--fill** option is requested: in this case, the GTK+ line wrap mode is automatically enforced.

  Note also that the **--fill** option will **left** justify any line containing an explicit line feed.



These options also affect the labels used in 2inputsbox, 3inputsbox, 2rangesbox and 3rangesbox widgets above each text entry/horizontal slider. The **alignement** of these labels is also affected by **--left** and **--right**, even if these labels are held on a single line.





- **--no-wrap** | **--wrap**

  When **--wrap** is force, Xdialog automatically wraps the backtitle and text around so to make them fit the widget width (note that as of v2.0.0, Xdialog uses its own wrapping routine as a replacement for the buggy GTK+ built-in one: but this does not apply when the --fill option is in force, where the GTK+ wrapping routine must be used...). The default is **--no-wrap** but remember that using the --fill options will enforce the (GTK+) wrap mode as well.

- **--cr-wrap** | **--no-cr-wrap**

  When passing a string parameter to Xdialog, Xdialog takes embeded newline characters into account by default (**--cr-wrap** is the default). Specifying **--no-cr-wrap** will prevent these newline characters to be taken into account for <text>, <backtitle>, <help> and --check option <label> parameters (thus allowing to split these strings into your script without having to escape the end of each line with a backslash). Note that whatever option is in force, any embeded "\n" characters sequence is always translated into a newline.

- **--stderr** | **--stdout**

  The default behaviour of Xdialog is to mimic (c) dialog and send all its results to stderr. This may sometimes be inapropriate and you may want to get separate outputs on separate streams (one for the true errors/warnings and one for the results). This is why the **--stdout** common option was implemented: it makes Xdialog to output all the widgets results to stdout instead of stderr. The **--stderr** common option is there so that you can revert to the default stderr output in a chained dialogs command line.

- **--separator** <character> | **--separate-output**

  The **--separator** option allows to change the separator used by the Xdialog widgets that return more than one result (2inputsbox, 3inputsbox, 2rangesbox, 3rangesbox, 2spinsbox, 3spinsbox, checklist, buildlist). The default separator is "/" but it may be unpractical to use such a separator. You can therefore change the separator to any character (examples: "\n" (line feed) or "|").

  The **--separate-output** is a (c) dialog compatible option and is synonym to **--separator "\n"**.

  Note: former Xdialog releases used the "\n" (line feed) as a results separator for the checklist widget; this has been changed to "/" in Xdialog v1.5.0 so to make it compatible with (c)dialog. In your old scripts using the Xdialog checklist, you will then have to add the **--separate-output** option before the **--checklist** one.

- **--buttons-style** default|icon|text

  The **--buttons-style** option must be followed by a parameter (the style name, case sensitive !) which must be either "default" (both icon and text in each button), "icon" (icon only in each button) or "text" (text only in each button).

Xdialog documentation - Box options

General notes about the box options parameters:

- The first parameter is a text string or a filename (this depends on the box option).

  The **<text>** parameter may contain "\n" characters sequences so that a long text can be splitted into several lines (see also the --wrap, --no-wrap, --cr-wrap and --no-cr-wrap common options about line wraping). Example:

  Xdialog --msgbox "text splitting\ntest..." 0 0

  The **<file>** parameter may be replaced, for all but the fselect and dselect widgets, by "-" so that the input stream is redirected to Xdialog stdin. Example:

  echo "hello folks" | Xdialog --textbox "-" 0 0

- Next come the menu **<height>** and **<width>** in characters (because the proportional fonts got variable characters width, the width taken into account is the average width of 0-9, A-Z and a-z). See also the --begin transient option regarding Xdialog widgets origin.

  These two numbers may be replaced by a single **<XSIZExYSIZE>** parameter (like the one passed in the -geometry option of X) which will represent the size of the Xdialog window in pixels (example: 300x200). As of v2.0.0, Xdialog also takes into account the position passed into the -geometry like parameter (e.g. 300x200+100-20). Note though that the actual Xdialog window position may be overidden by the window manager...

  Moreover, the Xdialog widgets may be auto-sized (by GTK+) by passing a 0 0 (or 0x0) size parameter to Xdialog. A position may also be passed to an auto-sizing Xdialog window (e.g.: 0x0+200+100) but you should avoid passing a negative offset in this case (e.g. 0x0-200+100) because Xdialog can't guess what size GTK+ will give to its window and can't therefore place the right (or bottom) side of its window relatively to the right (or bottom) screen edge.

  Last but not least, passing a -1 -1 (or -1x-1) size specification to Xdialog will make it maximize its window so to fill the whole screen (the origin also being set to 0 0).

  Examples:

```
Xdialog --msgbox "test" 6 20          # height of 6 characters and width of 20 characters
Xdialog --msgbox "test" 160x96        # width of 160 pixels and height of 96 pixels
Xdialog --msgbox "test" 160x96+200-100  # 160x96, left side at 200 pixels from the left
                                #  screen edge, bottom at 100 pixels from the bottom
                                #  of the screen
Xdialog --msgbox "test" 0 0           # auto-sized widget
Xdialog --msgbox "test" 0x0           # auto-sized widget
Xdialog --msgbox "test" 0x0+200+100   # auto-sized widget at origin 200x100 from the top
                                #  left corner of the screen.

Xdialog --msgbox "test" -1 -1         # maximized widget
Xdialog --msgbox "test" -1x-1         # maximized widget
```

- The other parameters are discussed into each box option description below.

Available box options and parameters:

- --yesno <text> <height> <width>
- --msgbox <text> <height> <width>
- --infobox <text> <height> <width> [<timeout>]
- --gauge <text> <height> <width> [<percent>]
- --progress <text> <height> <width> [<maxdots> [[-]<msglen>]]
- --inputbox <text> <height> <width> [<init>]
- --2inputsbox <text> <height> <width> <label1> <init1> <label2> <init2>
- --3inputsbox <text> <height> <width> <label1> <init1> <label2> <init2> <label3> <init3>
- --combobox <text> <height> <width> <item1> ... <itemN>
- --rangebox <text> <height> <width> <min value> <max value> [<default value>]
- --2rangesbox <text> <height> <width> <label1> <min1> <max1> <def1> <label2> <min2> <max2> <def2>
- --3rangesbox <text> <height> <width> <label1> <min1> <max1> <def1> <label2> <min2> <max2> <def2> <label3> <min3> <max3> <def3>
- --spinbox <text> <height> <width> <min> <max> <def> <label>
- --2spinsbox <text> <height> <width> <min1> <max1> <def1> <label1> <min2> <max2> <def2> <label2>
- --3spinsbox <text> <height> <width> <text> <height> <width> <min1> <max1> <def1> <label1> <min2> <max2> <def2> <label2> <min3> <max3> <def3> <label3>
- --textbox <file> <height> <width>
- --editbox <file> <height> <width>
- --tailbox <file> <height> <width>
- --logbox <file> <height> <width>
- --menubox <text> <height> <width> <menu height> <tag1> <item1> {<help1>}...
- --checklist <text> <height> <width> <list height> <tag1> <item1> <status1> {<help1>}...
- --radiolist <text> <height> <width> <list height> <tag1> <item1> <status1> {<help1>}...
- --buildlist <text> <height> <width> <list height> <tag1> <item1> <status1> {<help1>}...
- --treeview <text> <height> <width> <list height> <tag1> <item1> <status1> <item_depth1> {<help1>}...

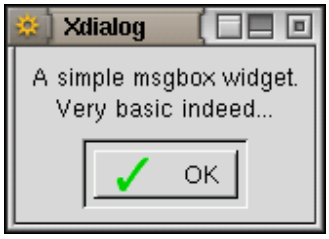
-
- --fselect <file> <height> <width>
-
- --dselect <directory> <height> <width>
-
- --calendar <text> <height> <width> <day> <month> <year>
-
- --timebox <text> <height> <width>
-

- **--yesno** <text> <height> <width>
- **--msgbox** <text> <height> <width>

These widgets just display the <text> and wait for the user to press a button. The buttons in the **yesno** widget are (surprise, surprise !) *Yes* and *No*, while a single *OK* button is used for the **msgbox**.

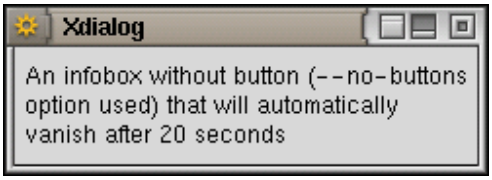
The **yesno** widget accepts the --wizard transient option (that may itself be used together with --no-cancel); in this case the *Yes* and *No* buttons are replaced by the *Previous*, *Cancel* (if --no-cancel is not in force) and *Next* buttons. The **yesno** widget also accepts the --default-no transient option.

Both widgets accept the --icon, --help and --check transient options.



- **--infobox** <text> <height> <width> [<timeout>]

The infobox displays a <text> message and accepts an optional <timeout> parameter (the default is 1000). The infobox vanishes after **<timeout>/1000** seconds or when the *OK* button is pressed (this button may be suppressed by using the --no-buttons transient option).



If a 0 timeout is passed as parameter, then the infobox widget behaves differently:

  - The *OK* button is replaced by a *Cancel* button (unless the --no-buttons transient option is in force).
  - It accepts messages from stdin (much in the same way as the gauge widget). These messages may be "XXXX" (in which case the infobox is closed) or new <text> enclosed by "XXX" markers; newlines can be inserted into the <text> (in the same way as for the gauge widget).

- It waits until the *Cancel* button is pressed, or the "XXXX" message is received, or until the stdin is put at EOF (unless the --ignore-eof transient option is in force).

This widget also accepts the --icon transient option.

- **--gauge** <text> <height> <width> [<percent>]

  This widgets displays the <text> and a progress bar (the gauge) which starting position is set according to the <percent> parameter (if any, the default being 0%).

  Once set up, it accepts new percentage values (the gauge being updated accordingly) on stdin as well as new <text> enclosed by two "XXX" markers; a newline can be inserted into the new <text> by issuing:

  echo "\\n"

  between each line sent to Xdialog stdin (i.e. Xdialog must actually receive on stdin a string holding the two "\" and "n" characters, and not just a line feed).

  The **gauge** widget vanishes once the percent value exceeds 100% or when its stdin is put at EOF (unless the --ignore-eof transient option is in force).

  This widget also accepts the --icon transient option.

  See also the (c)dialog compatibility notes.

- **--progress** <text> <height> <width> [<maxdots> [[-]<msglen>]]

  The **progress** widget looks exactly as a gauge widget but behaves differently. It is designed so that it can interact easily with console utilities issuing progress reports on their output stream, either in the form of "dots" (or hashes, stars, etc...) or of a number (percentage or any counter).

  The <text> parameter is displayed and may be appended with a message which is read from Xdialog stdin and which number of characters is <msglen>. If you don't care about the leading message sent by the console utility before the actual "dots"/values are sent, then you may pass the number of characters to be ignored as a negative value. Unlike the gauge widget, there is no way to change this text once it is setup and displayed. When <msglen> is omitted (or when <msglen> is 0), none of the leading message characters (if any) are appended to <text> nor ignored (beware that in this case, any leading message characters will be taken into account as "dots" !).

  The progress bar is initially set to 0 and will accept any number of "dots" or any counter value below <maxdots>. Each time a new "dot" or a new value is received on the Xdialog input stream (stdin), the progress bar is updated accordingly (a percentage is also calculated and shown into the progress bar). If <maxdots> is omitted or passed as 0, then a default max value of 100 is used.

  The widget is automatically closed whenever the <maxdots> number is exceeded or when Xdialog stdin is put at EOF.

Here are two examples of how to use this widget: allrpms and format1440.

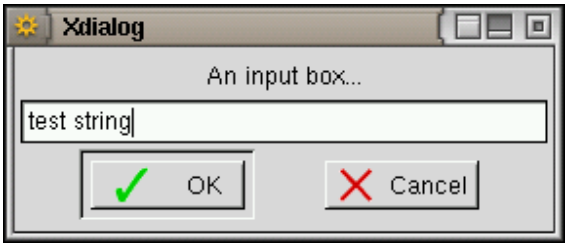
This widget also accepts the --icon transient option.

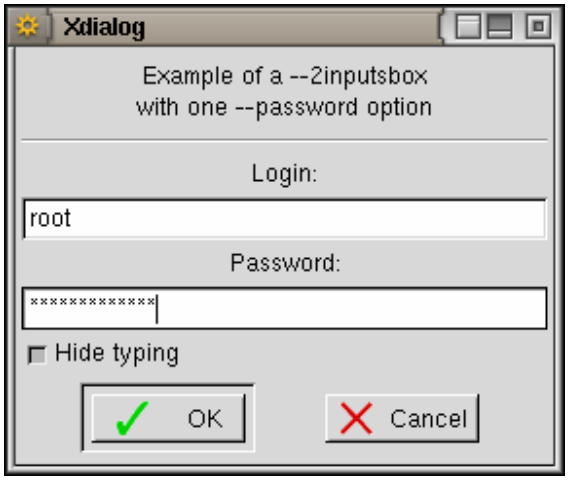
- **--inputbox** <text> <height> <width> [<init>]
- **--2inputsbox** <text> <height> <width> <label1> <init1> <label2> <init2>
- **--3inputsbox** <text> <height> <width> <label1> <init1> <label2> <init2> <label3> <init3>

The **--inputbox** widget displays a <text> together with an entry field accepting any string. The string typing into the field may be hidden (entered characters are displayed as "*") thanks to the --password transient option. The entered string is returned (printed to Xdialog output stream) when the *OK* button is pressed (nothing is returned if the *Cancel* button is pressed; this button may itself be removed from the widget by using the --no-cancel transient option). An optional <init> string may be passed so to be setup as the default entered string (which will appear into the text entry field when the widget is drawn).

The **--2inputsbox** and the **--3inputsbox** widgets allow for two or three entry fields into the same box (see the --password transient option to learn how these fields are affected). A <label> is setup above each field (see the --center, --left, --right and --fill common options to learn how the labels justification and alignement can be affected); as for the <text> parameter, the <label>s may hold "\n" sequences IOT force text splitting into several lines. The <init> strings cannot be omitted but may perfectly be NULL (empty) strings.

All three widgets also accept the --interval, --icon, --no-buttons, --default-no, --wizard, --help and --check transient options.





- **--combobox** <text> <height> <width> <item1> ... <itemN>

The combobox displays a <text> together with an entry field to which a pull-down list of <items> is attached: the user may choose an item into the pull-down list or edit the entry field (provided the --editable transient option was specified). Xdialog returns the entry field contents once the *OK* button is pressed.

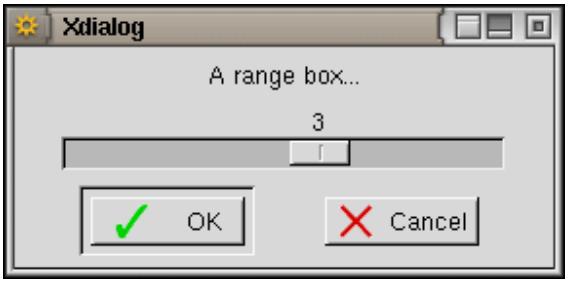
This widget also accepts the --interval, --icon, --no-buttons, --default-no, --wizard, --help and --check transient options.

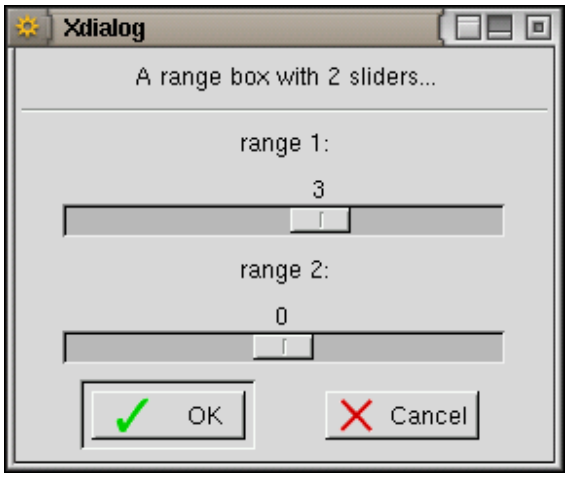
- **--rangebox** <text> <height> <width> <min value> <max value> [<default value>]
- **--2rangesbox** <text> <height> <width> <label1> <min1> <max1> <def1> <label2> <min2> <max2> <def2>
- **--3rangesbox** <text> <height> <width> <label1> <min1> <max1> <def1> <label2> <min2> <max2> <def2> <label3> <min3> <max3> <def3>

The **--rangebox** widget displays a <text> together with a horizontal slider which scale ranges from <min value> to <max value>. The initial position of the cursor on the slider is set to <default value> (when the <default value> parameter is omitted, the cursor position is set to the <min value>). The value corresponding to the current cursor position is returned (printed to Xdialog output stream) when the *OK* button is pressed (nothing is returned if the *Cancel* button is pressed; this button may itself be removed from the widget by using the --no-cancel transient option).

The **--2rangesbox** and the **--3rangesbox** widgets allow for two or three sliders into the same box. A <label> is setup above each slider (see the --center, --left, --right and --fill common options to learn how the labels justification and alignement can be affected); as for the <text> parameter, the <label>s may hold "\n" sequences IOT force text splitting into several lines. The <def> values cannot be omitted.

All three widgets also accept the --interval, --icon, --default-no, --wizard, --help and --check transient options.



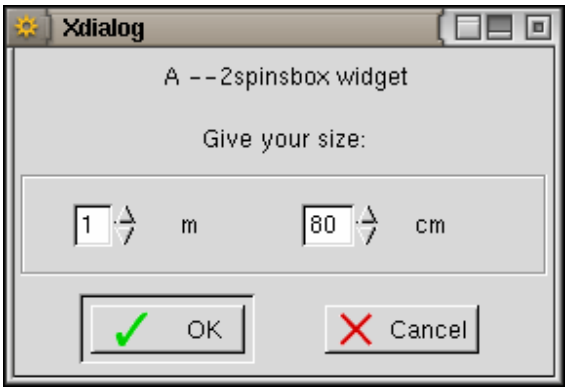


- **--spinbox** <text> <height> <width> <min> <max> <def> <label>
- **--2spinsbox** <text> <height> <width> <min1> <max1> <def1> <label1> <min2> <max2> <def2> <label2>
- **--3spinsbox** <text> <height> <width> <min1> <max1> <def1> <label1> <min2> <max2> <def2> <label2> <min3> <max3> <def3> <label3>

These widgets are more suited than ranges boxes for input of values with units. They display a <text> and one to three spin buttons which value may range from <min> to <max> and defaults to

<def>. Each spin box is followed by a <label> (more likely a unit name or a separator); this label is not setup if passed as an empty string.

All three widgets accept the --interval, --icon, --default-no, --wizard, --help and --check transient options.

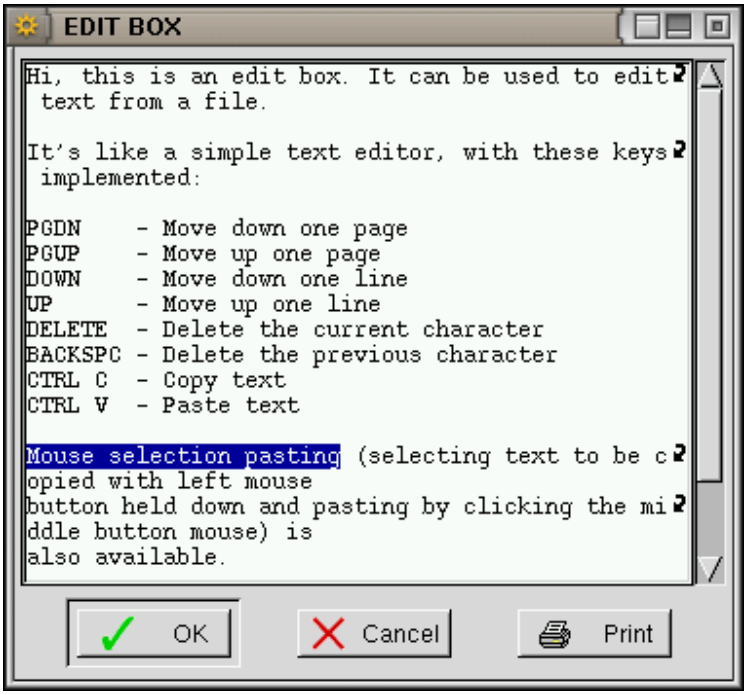


- **--textbox** <file> <height> <width>
- **--editbox** <file> <height> <width>

These widgets allow to display a text file contents. If the <file> parameter is replaced with a "-" (minus sign), then the text to be displayed is read from Xdialog stdin. The **--editbox** allows to edit the text and returns it (i.e. prints it on Xdialog output stream) once the *OK* button is pressed.

Both widgets accept the --help, --default-no, --no-cancel, --fixed-font, --print, --wizard and --check transient options. The **textbox** also accepts the --no-buttons transient option.

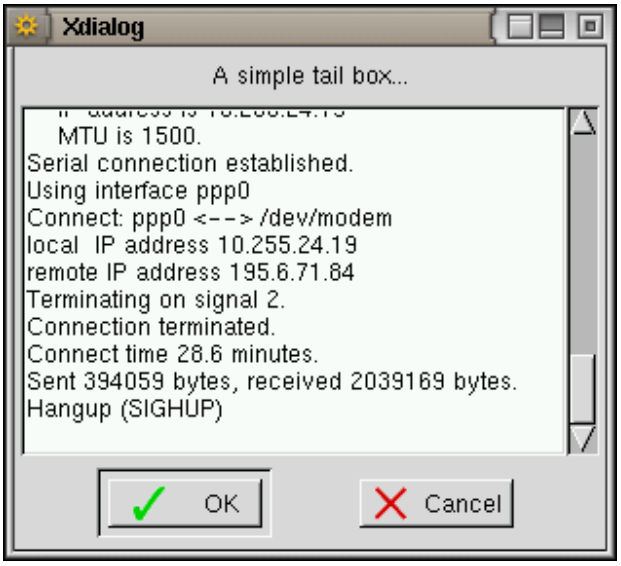
See also the (c)dialog compatibility notes.



- **--tailbox** <file> <height> <width>

The **tailbox** widget ressembles to a textbox but the text is automatically scrolled to the end and is regularly updated (so that any **addition** to the file is reflected into the tailbox; note that if the file is **truncated**, **deleted** or **overwritten** by another program while the tailbox is displaying it, the update is stopped). As for the textbox, if the <file> parameter is replaced with a "-" (minus sign), then the text to be displayed is read from Xdialog stdin.

This widget accepts the --smooth, --help, --default-no, --no-buttons, --no-ok, --no-cancel, --fixed-font, --print, --wizard and --check transient options.



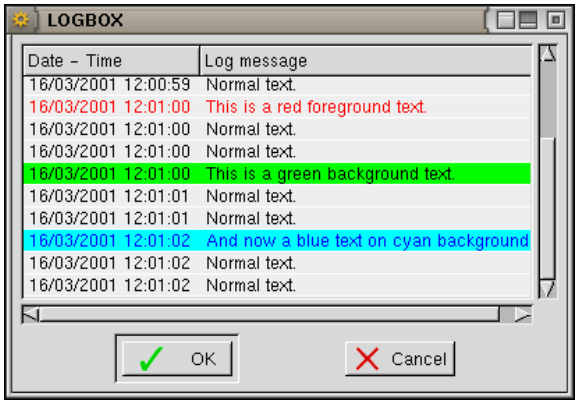
- **--logbox** <file> <height> <width>

The **logbox** is much like a tailbox but it may use different colours (both foreground and background) for each line it displays. The messages (i.e. each line) may as well be time/date stamped thanks to the --time-stamp and --date-stamp transient options. Moreover the messages may appear in reverse order (i.e. last message at the top of the list) thanks to the --reverse transient option.

On the other hand, the log box cannot use the --fixed-font and --print transient options, the text is not wrapped automatically when the box is not large enough for lines to fit in it (but you may use the horizontal scroll bar to view the whole lines), and the text displayed can't be selected with the mouse (for copy and paste purpose).

Colour selection is controlled by the insertion of escape sequences into the text held in <file>. Recognized escape sequences are **ESC[1;30m** to **ESC[1;38m** (for foreground colour) and **ESC[1;40m** to **ESC[1;48m** (for background colour), any combination of foreground and background colours being accepted as well (e.g. **ESC[1;31;43m**). Note that only the first escape sequence encountered in each line is taken into account, and whatever is its actual position into the line (at the start of the line, embedded in the text, or at the end of the line), the whole line will be given the corresponding attribute(s) (i.e. you can't highlight a single word into a line). By using the --keep-colors transient option, you can instruct Xdialog to keep the foreground and background colour setting from one message to the others (i.e. until a new escape sequence is received).

This widget also accepts the --smooth, --help, --default-no, --no-buttons, --no-ok, --no-cancel, --wizard and --check transient options.



COLOUR	FOREGROUND CODE	BACKGROUND CODE
BLACK	30	40
RED	31	41
GREEN	32	42
YELLOW	33	43
BLUE	34	44
MAGENTA	35	45
CYAN	36	46
WHITE	37	47
GTK+ theme settings	38	48

- **--menubox** <text> <height> <width> <menu height> <tag1> <item1> {<help1>}...

  This widget presents the <text> together with a menu. For each line in the menu, there must be a <tag> and an <item> parameters. The tag is displayed left (provided that the --no-tags transient option is not in force) and the menu item right.

  The <help> parameters are only to be used if the --item-help transient option is in force; the help text is then displayed into a status bar below the menu each time a new menu item is selected.

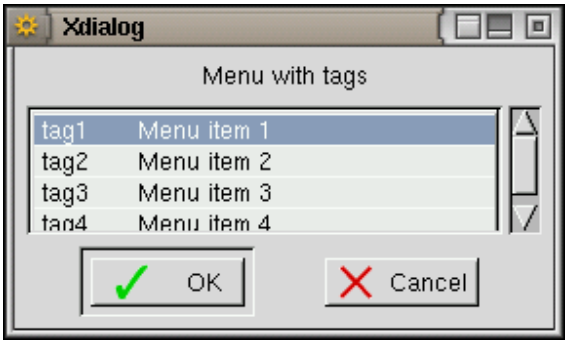
  The <menu height> is the number of lines (in characters) to which the menu sub-window should be set (a height of 0 will let Xdialog auto-size); of course the overall widget <height> must be big enough (or the widget auto-size must be in force), else the <menu height> will not be taken into account.

  The default selected row in the menu is normally the first row, but this can be changed thanks to the --default-item transient option.

  Although there is no <status> parameter for the menubox, it is possible to make a menu item unavailable by setting its <tag> to an empty string (in this case, if the --no-tags transient option is not in force, the tag will appear as a tilde). The unavailable rows appear in dark grey text on light grey background.

  When the *OK* button is pressed, the widget prints, onto the Xdialog output stream, the tag corresponding to the last selected menu entry.

  This widget also accepts the --interval, --icon, --default-no, --wizard, --help and --check transient options.



- **--checklist** <text> <height> <width> <list height> <tag1> <item1> <status1> {<help1>}...
- **--radiolist** <text> <height> <width> <list height> <tag1> <item1> <status1> {<help1>}...

These widgets present the <text> together with a list of <item>s each one being prefixed with their corresponding <tag> (provided that the --no-tags transient option is not in force).

Each <status> parameter (which value may be "on", "off" or "unavailable") tells to Xdialog if the corresponding <item> must be selected as default ("on"), unset but available ("off") or "unavailable" (i.e. visible but not selectable); note that it is also possible to make an item unavailable by setting its <tag> to an empty string. Exactly one item is selected at any time in a **radiolist**, while the **checklist** allows for any number of items (0 to all items) to be selected at a time.

The <help> parameters are only to be used if the --item-help transient option is in force; the help text is then displayed as tooltips when the mouse pointer stays long enough (usually 0.5s) over an item.

The <list height> is the number of lines (in characters) to which the list sub-window should be set (a height of 0 will let Xdialog auto-size); of course the overall widget <height> must be big enough (or the widget auto-size must be in force), else the <list height> will not be taken into account.

The tag(s) of the selected item(s) are sent onto the Xdialog output stream when the *OK* button is pressed. For the **checklist**, each tag is separated from the other with a "/" character; this separator may be changed by using either the --separator or the --separate-output common options.

These widgets also accept the --interval, --icon, --default-no, --wizard, --help and --check transient options.

- **--buildlist** <text> <height> <width> <list height> <tag1> <item1> <status1> {<help1>}...

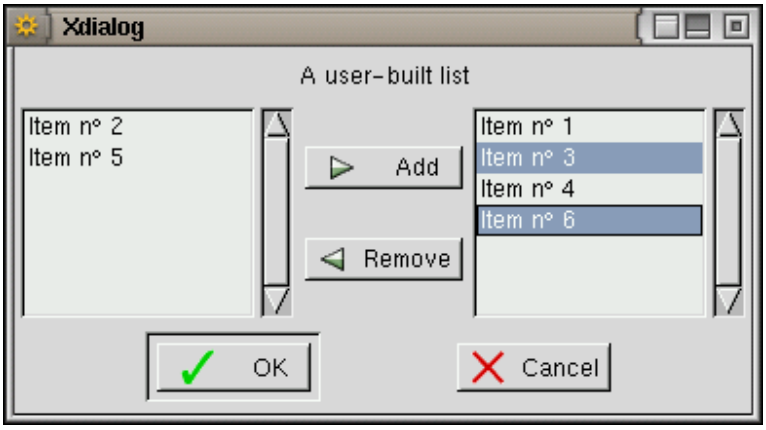
This widget presents the <text> together with two list sub-windows and two buttons (*Add* and *Remove*). In the left-most sub-window all unselected and unavailable items are listed (those items either had an "off" or "unavailable" <status> or have been *Remove*d from the list by the user). The right-most sub-window presents the list of all selected items (those items either had an "on" <status> or have been *Add*ed to the list by the user). Note that it is also possible to make an item unavailable by setting its <tag> to an empty string.

The <help> parameters are only to be used if the --item-help transient option is in force; the help text is then displayed as tooltips when the mouse pointer stays long enough (usually 0.5s) over an item.

The <list height> is the number of lines (in characters) to which the list sub-window should be set (a height of 0 will let Xdialog auto-size); of course the overall widget <height> must be big enough (or the widget auto-size must be in force), else the <list height> will not be taken into account.

IOT *Add* or *Remove* items, the user must first highlight them and then press the proper button. The items appear in the order in which they were added/removed and this order is kept when the result is sent to the Xdialog output stream (when the *OK* button is pressed): the corresponding <tag>s are sent in the exact order in which the associated items appear in the right-most sub-window. Each tag is separated from the other with a "/" character; this separator may be changed by using either the --separator or the --separate-output common options.

This widget also accepts the --interval, --icon, --default-no, --wizard, --help and --check transient options.



- **--treeview** <text> <height> <width> <list height> <tag1> <item1> <status1> <item_depth1> {<help1>}...

  This widget presents the <text> together with a sub-window holding a tree of the <item>s in the order in which they appear into the Xdialog command line and at the associated <item_depth> (which should range from 1 to 24). Note that from one item to the next, the depth must never increase by more than one while it may decrease by more than one. The <list height> is the number of lines (in characters) to which the list sub-window should be set (a height of 0 will let Xdialog auto-size); of course the overall widget <height> must be big enough (or the widget auto-size must be in force), else the <list height> will not be taken into account. When the *OK* button is pressed, the widget sends the <tag> associated with the last selected item to the Xdialog output stream.

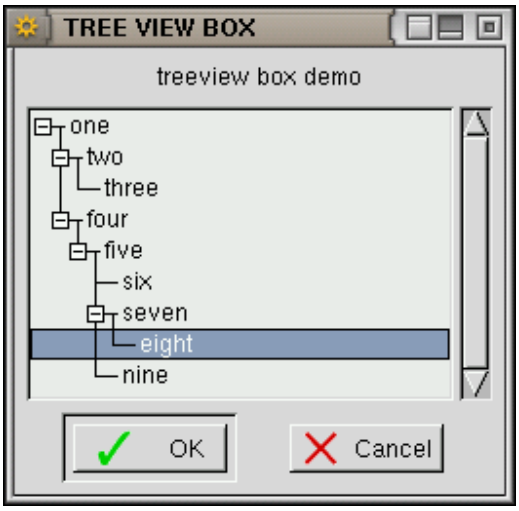
  Each <status> parameter (which value may be "on", "off" or "unavailable") tells to Xdialog if the corresponding <item> must be selected as default ("on"), unset but available ("off") or "unavailable" (i.e. visible but not selectable); it is also possible to make an item unavailable by

setting its <tag> to an empty string. Note that if an item with an "unavailable" <status> is the root of a branch of the tree, then all the items in this branch can't be accessed (whatever is their own <status>). Exactly one item is selected at any time.

The <help> parameters are only to be used if the --item-help transient option is in force; the help text is then displayed as tooltips when the mouse pointer stays long enough (usually 0.5s) over an item.

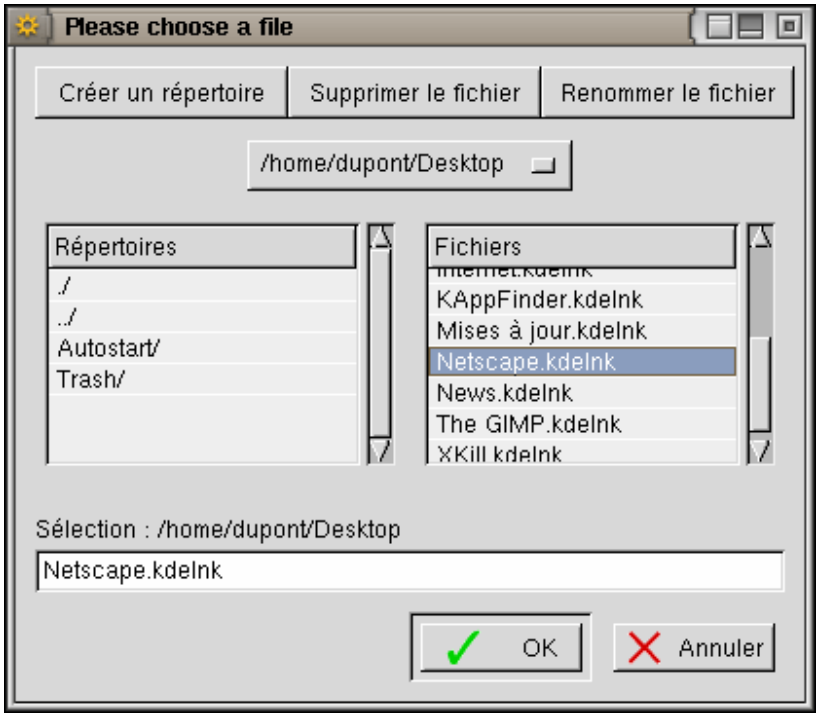
This widget also accepts the --interval, --icon, --default-no, --wizard, --help and --check transient options.

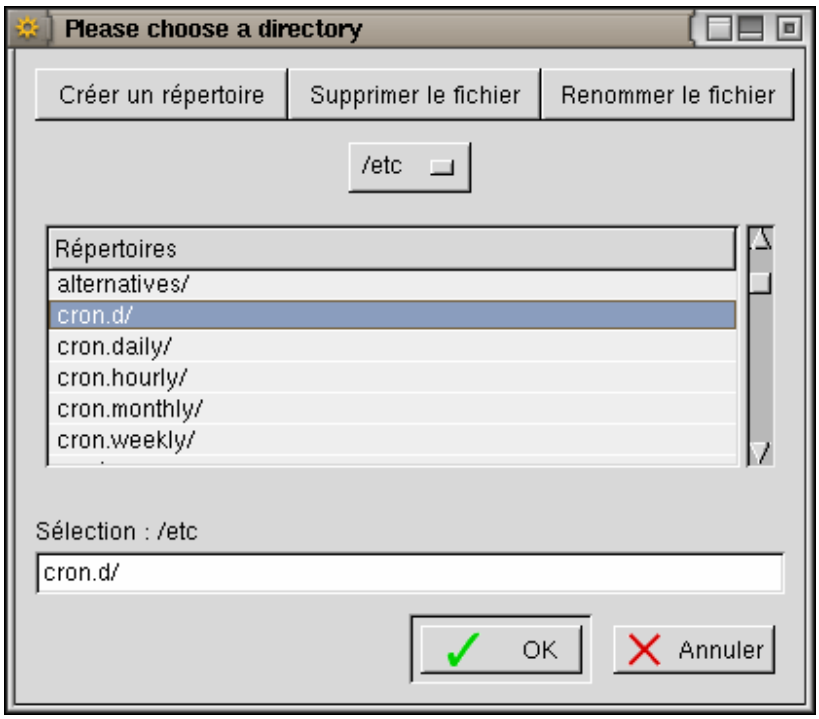
Note: there is a bug in all GTK+ releases (up to and including v1.2.10) that prevents Xdialog to highlight the default selected item (the "on" <status> of this item is actually taken into account but the item can't be highlighted when the widget is drawn).



- **--fselect** <file> <height> <width>
- **--dselect** <directory> <height> <width>

These widgets make use of the GTK+ file selector. They allow to select a filename (for the fselect widget) or a directory name (for the dselect widget). The <file> (for fselect) and <directory> (for dselect) parameters are used as the default selection. Using the --no-buttons transient option prevents the *New directory*, *Delete file* and *Rename file* buttons to be displayed in the file selector. The --help, --wizard, --no-cancel and --default-no transient options are allowed, even if --no-buttons was specified. The --check transient option may also be used. The widgets return the user-selected file/directory name once the *OK* button is pressed.

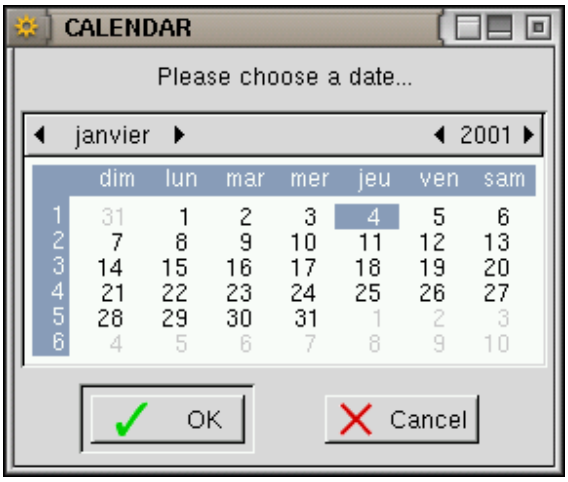




- **--calendar** <text> <height> <width> <day> <month> <year>

  This widget displays the <text> together with a calendar showing the month holding the date defined by the <day> (1 to 31) <month>, (1 to 12) and <year> (1970 and over) parameters (if any of this parameter is 0, then the current date is used). The user may then browse the calendar and choose another date. Once the *OK* button is pressed, the widget returns the user-selected date (in the form DD/MM/YYYY).

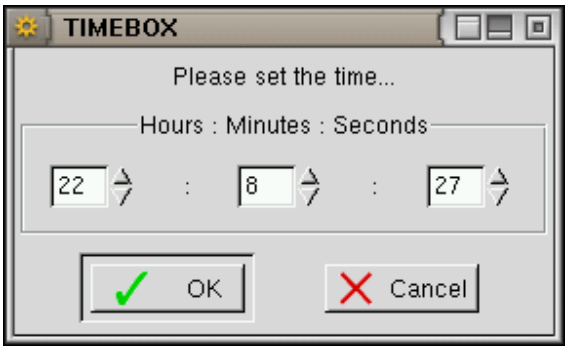
This widget also accepts the --interval, --icon, --default-no, --wizard, --help and --check transient options.



- **--timebox** <text> <height> <width>

  This widget displays the <text> together with the current time into three "spin buttons" (hours, minutes and seconds). The user may change the time displayed and the widget returns the user-set time (in the form HH:MM:SS) once the *OK* button is pressed.

  This widget also accepts the --interval, --icon, --default-no, --wizard, --help and --check transient options.



Xdialog documentation - Transient options

The **transient options** only apply to the next box option into the Xdialog command line.

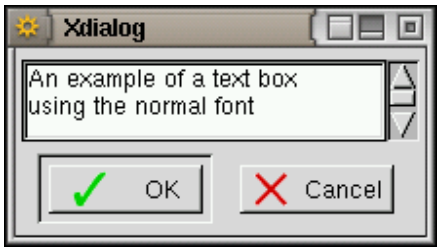
Available transient options and parameters:

- --fixed-font
-
- --password

-
- --editable
-
- --time-stamp | --date-stamp
-
- --reverse
-
- --keep-colors
-
- --interval <timeout>
-
- --no-tags
-
- --item-help
-
- --default-item <tag>
-
- --icon <xpm filename>
-
- --no-ok
-
- --no-cancel
-
- --no-buttons
-
- --default-no
-
- --wizard
-
- --help <help>
-
- --print <printer>
-
- --check <label>
-
- --ok-label <label>
-
- --cancel-label <label>
-
- --beep
-
- --beep-after
-
- --begin <Yorg> <Xorg>
-
- --ignore-eof
-
- --smooth
-

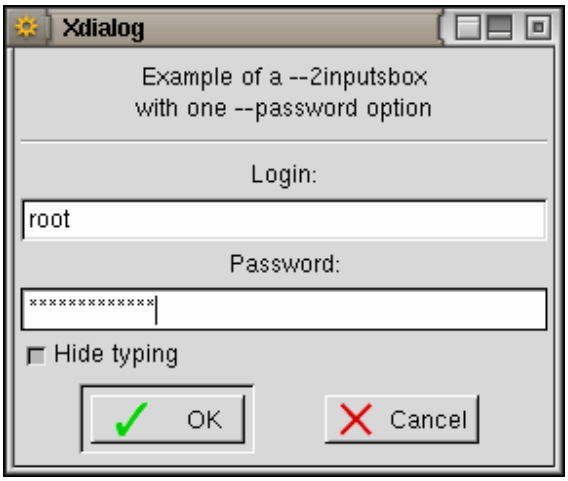
- **--fixed-font**

This option allows to use a monospacing font in the text area of tailbox, textbox and editbox widgets.



- **--password**

  This option is for use with input(s) box widgets (--inputbox, --2inputsbox and --3inputsbox box options). Its effect is to turn the last input field of an inputbox widget into a password entry field (in which every character typed is displayed as a "*"). It is possible to repeat this option twice or three times before --2inputsbox and --3inputsbox so that the last two or the three input fields are turned into password input fields. As of Xdialog v2.0.6, a check button (*Hide typing*) is also setup when this option is in use: it allows to toggle the hiding of the password fields (hiding being on by default).



- **--editable**

  This option is for use with the combobox widget. Its effect is to allow the user to edit the entry field of the combo box (by default it is not allowed to do so: only one of the items into the combo box pull-down list may be choosed).

- **--time-stamp** | **--date-stamp**

  These options make a second column to appear on the left of the messages displayed by the logbox widget. This column is used to display the time (with **--time-stamp**) or date plus time (with **--date-stamp**) at which each message is received by the logbox. If either of these two options is used, then the column titles ("Time stamp" or "Date - Time", and "Log messages") also appear at the top of the logbox.

- **--reverse**

This option makes the messages displayed in the logbox widget to appear in reverse order (the last received message being displayed at the top of the messages list).

- **--keep-colors**

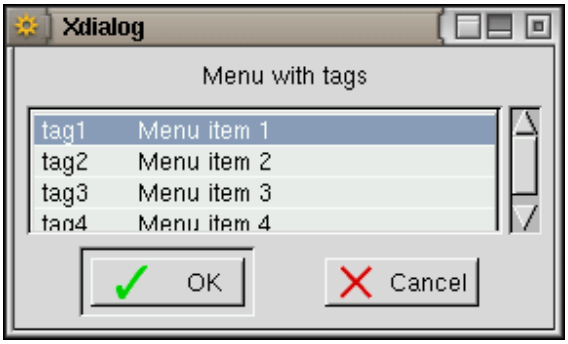
  This option is for use with the logbox widget and instructs Xdialog to remember the last foreground and background colours message line setting so to use it in the next lines until a new colour specification (escape sequence) is received.

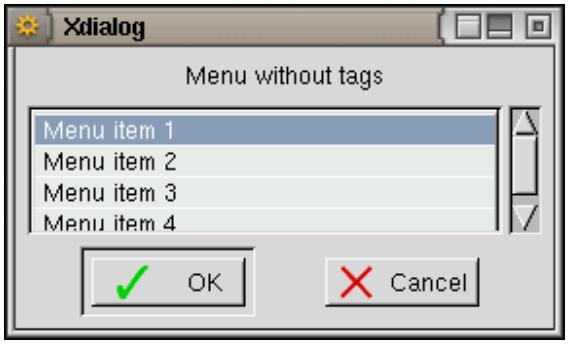
- **--interval** <timeout>

  This option will make most Xdialog widgets that return results (input(s) boxes, combo box, range(s) boxes, spin(s) boxes, list boxes, menu box, treeview, calendar, timebox) to report regularly these results on Xdialog output stream until the user chooses/enters the definitive result or the widget is closed. As an example, a rangebox can be made to report its current cursor position every 2s. The timeout parameter is in milli-seconds (it must be positive; a zero timeout will make this option to be ignored).

- **--no-tags**

  This option is for use with menubox, checklist and radiolist widgets. Each menu/list item in these widgets is normally preceeded with its <tag>; when using this option, the tags are not displayed.





- **--item-help**

  This option is for use with the menubox, checklist, radiolist, buildlist and treeview widgets. It makes these widgets to accept an additional <help> parameter for each item; this parameter is a text string that will be displayed as a tooltip (for checklist, radiolist, buildlist and treeview widgets) when the mouse pointers stays for some time (usaually 0.5s) over an item, or into a status bar (below the menu window of the menubox widget) when an item is selected.

- **--default-item** <tag>

  This option is for use with the menubox and allows to select (and move to) a given default row (which tag is <tag>).

- **--icon** <xpm filename>

  This option must be followed by the filename of an icon (in XPM format only). This icon will be displayed on the left of the <text> (provided the following box option accepts such a <text> parameter, which is **not** the case of the textbox, editbox, tailbox, logbox, fselect and dselect widgets). If the filename does not corresponds to a XPM image, the option will be ignored.

- **--no-ok**

  This option allows to suppress the *OK* button from the tailbox and logbox widgets.

- **--no-cancel**

  This option allows to suppress the *Cancel* button from all but the infobox, gauge and progress widgets.

- **--no-buttons**

  This option allows to suppress all the *OK*, the *Cancel*, the *Help* and the *Print* buttons from the textbox, tailbox, logbox, infobox widgets, as well as the *New directory*, *Delete file* and *Rename file* buttons from the fselect and dselect widgets.

- **--default-no**

  Sets the *No* or *Cancel* button to be the default (pre-selected) one. This option has no effect when used with the --wizard option.

  See also the (c)dialog compatibility notes.

- **--wizard**

  This option turns Xdialog widgets into "installation wizard" widgets by replacing the *Yes* or *OK* button with *Next*, the *No* button with *Cancel* (the *Cancel* button is not displayed if the --no-cancel option is in force), and by adding a *Previous* button (which makes the widget to return an exit code of 3 when pressed). This option may apply to all widgets but the msgbox, infobox, gauge and progress widgets.

- **--help** <help>

  This option makes a *Help* button to appear in all but the infobox, gauge and progress widgets, and is to be followed by the help text that will be displayed (in a msgbox) when the *Help* button is chosen. The help box is set to the same size as the widget from which it was invoked (it is auto-sized if this widget was itself auto-sized). The <help> text may hold "\n" (as for the <text>

parameter) so to force help text line splitting. If the help text is an empty string then, when the *Help* button is pressed, the widget is closed and an exit code of 2 is returned.

- **--print** <printer>

  Makes a *Print* button to appear in textbox, editbox and tailbox widgets (provided that the --no-button option was not specified). Hitting the button will make Xdialog to issue a printing command defined at compile time, defaulting to:

  lpr -P<printer> /tmp/Xdialog.tmp

  If the <printer> parameter is an empty string, then the -P option is not used and the issued command is:

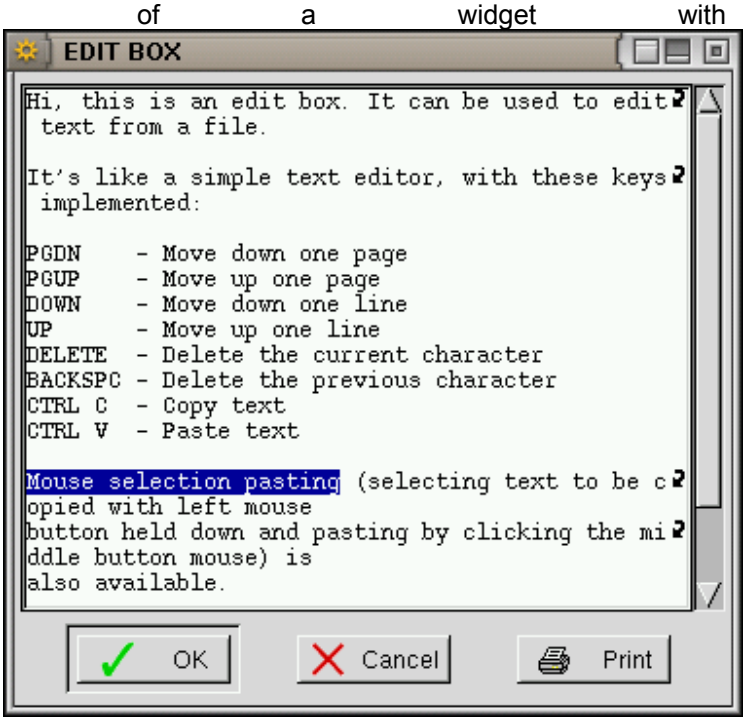
  lpr /tmp/Xdialog.tmp

  The <printer> parameter (that must follow the --print option) gives the name of the printer to use. The /tmp/Xdialog.tmp file is a temporary file used by Xdialog and is deleted when Xdialog terminates. To see what printing command is used by your version of Xdialog, just type:

  Xdialog

  (without parameter) and look at the last lines in the displayed usage.

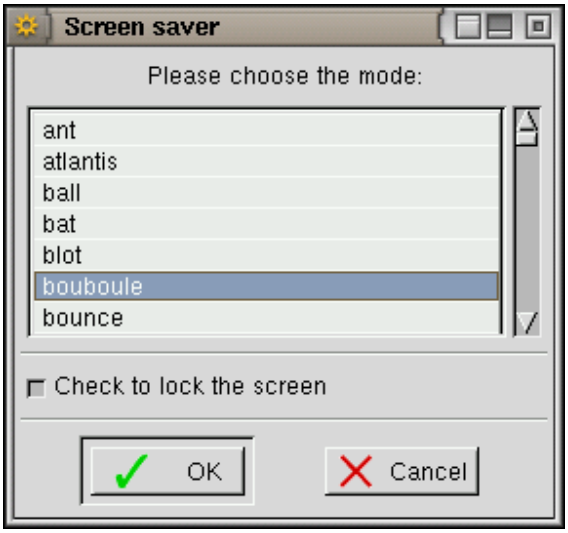
  Example of a widget with a *Print* button:



- **--check** <label>

  This option applies to all but the infobox, gauge and progress widgets. It adds a check button (on the left) and a label (on the right) at the bottom of the widget and makes Xdialog to report either "checked" or "unchecked" on its output stream (as the last string for widgets that return strings by themselves) when then *OK* button is pressed.

The <label> may contain "\n" characters sequences so to force line breaks (it is not possible to auto-wrap this label) but the text justification is not affected by any Xdialog justification option in force.



- **--ok-label** <label>
- **--cancel-label** <label>

  These options allow to change the labels of (respectively) *OK* or *Yes* buttons, and *Cancel* or *No* buttons in any Xdialog widget using these buttons. These options are ignored if the --wizard transient option is in force.

- **--beep**
- **--beep-after**

  These options instruct Xdialog to emit a short beep either just before displaying the next widget (**--beep**) or just after closing it (**--beep-after**). Both options may be used together and apply to any widget.

- **--begin** <Yorg> <Xorg>

  This option asks Xdialog to open its widget at an absolute position (in characters) on the screen. It will work only if your window manager does take the application requested window position into account (many ignore it or may be configured so to ignore it). This option is for use when the widget size is given in characters as well (if you give it in pixels, then the **--begin** option is ignored; give a "-geometry"-like size/position specification instead. E.g.: 400x200+20-30). This option may be used with any widget.

- **--ignore-eof**

  This option is for use with the infobox and gauge widgets. It prevents Xdialog to close its widget when its input stream is put at EOF (use with care as you must ensure that a way remains to close the Xdialog widget !). This may be useful when piping data to Xdialog through a FIFO.

- **--smooth**

  Will give a smooth, flicker free (but significantly slower) scrolling in tailbox and logbox widget. For use in big windows displaying a slowly updated log...

Note: from Xdialog v2.0.4 onwards, the tailbox and logbox are, when passed the name of an actual file as parameter, loading all the current file contents in one go at startup before displaying it; while this is quite fast with the tailbox, it may take quite some time with the logbox (for which many more processing is performed for each line of text). The use of **--smooth** with the logbox therefore disables this feature (so that a display update is done each time a new line of text is loaded), thus reverting to the pre-v2.0.4 Xdialog versions behaviour.

Xdialog documentation - Special options

These options are "special" because they just make Xdialog to print a string on stderr and exit immediately without displaying any widget (they should therefore not be used together with any other option).

Available special options:

- --version
-
- --print-version
-
- --print-maxsize
-

- **--version**
- **--print-version**

  These options make Xdialog to print its version number to stderr and exit. This is useful to check from a script if the installed version of Xdialog is able to use some of the latest widgets that have been implemented since Xdialog exists... The difference between **--version** and **--print-version** is that the later prefixes the version number with "Version: " (in a cdialog compatible way).

  Examples:

  $ Xdialog --version
  1.5.1
  $ Xdialog --print-version
  Version: 1.5.1
  $ _

  See also the (c)dialog compatibility notes.

- **--print-maxsize**

This option (cdialog compatible) makes Xdialog to print the maximum possible size usable for the Xdialog widgets so that they fit the screen. The size is given in characters in the following form: MaxSize: <height>, <width>

Example:

```
$ Xdialog --print-maxsize
MaxSize: 62, 144
$ _
```

Xdialog documentation - GTK+ options

As most GTK+ programs, Xdialog does recognize the GTK+ options. Only a few of them are actually useful though (most of them dealing with GTK+ debugging facilities).

Available GTK+ options (1):

- --display <display>
- --name <name>
- --class <class>
- --sync
- --no-xshm
- --g-fatal-warnings
- --gtk-module <module>
- --gtk-debug <flags>
- --gtk-no-debug <flags>
- --gdk-debug <flags>
- --gdk-no-debug <flags>

(1) there may be more or less options available, depending on the GTK+ version, please refer to the GTK+ documentation for options actually available on your system.

- **--display** <display>

  This option allows to specify a X <display> on which the Xdialog window is to be open. This is useful when invoking Xdialog from a script executed by a daemon (e.g.: **atd** or **crond**) which does not set the **$DISPLAY** variable needed by GTK+.

- **--name** <name>
- **--class** <class>

  These options allow to set respectively the <name> and the <class> of a GTK+ program, for use by the window manager. Xdialog got its own --wmclass option that sets both its name and class to the same value. You may still want to use these two GTK+ options if you need to set different name and class values.

- **--sync**

  Tells GTK+ (actually the GDK part of GTK+) to make X calls synchronous. This will give very smooth display updates, but very slow as well... No practical use.

- **--no-xshm**

  Prevents GTK+ (actually the GDK part of GTK+) to use the X shared memory extension. In case your X server XSHM is broken (in this case, upgrade !)... No practical use.

- **--g-fatal-warnings**

  Make all GTK+ warnings fatal. Xdialog does not make such warnings to be issued (if it does, it is a bug, please report it !). But you may encounter GTK+ warnings while using Xdialog if your own GTK+ configuration is wrong (e.g. if you are using a GTK+ theme needing for a specific gtk-engine which is actually missing from your system); in this case you will also encounter the same warnings using any other GTK+ application. Using this option will make Xdialog to be killed when a warning is issued rather than reporting this warning on Xdialog stderr (and therefore possibly polluting the strings returned by Xdialog, when the --stderr option is in force). For debugging purpose, no practical use.

- **--gtk-module** <module>

  For additional module loading. Not needed by Xdialog.

- **--gtk-debug** <flags>
- **--gtk-no-debug** <flags>
- **--gdk-debug** <flags>
- **--gdk-no-debug** <flags>

  Set/unset GTK/GDK debugging <flags>. For debugging purpose only, no practical use.

Xdialog documentation - Compatibility

dialog-compatible utilities:

There are a few **dialog** "compatible" utilities around, some of them just mimic **dialog** without adding new features (e.g. **gdialog**), others add features and may differ in their syntax (**cdialog**, **wdialog**). I currently know about:

- **dialog**: the original one (up to v0.3) which has been upgraded and declined in further versions by contributors (v0.4, v0.6a-0.6z, v0.7). AFAIK it is no more actively developed.
- **cdialog**: a much improved version of dialog which is still developed. Its binary file may actually be named **cdialog** or **dialog**, but you may check the usage message printed for the "ComeOn Dialog!" string IOT find out if your "dialog" binary is in fact "cdialog". You may as well try:

  dialog --print-version

  which should report v0.9 or higher for cdialog.

- **gdialog**: a Gnome utility that mimics closely **dialog** but that misses the new features of **cdialog**.
- **wdialog** and **Wxdialog**: a set of utilities which aim is to provide an installation wizard for both UNIX console (using wdialog) and X11 (using Wxdialog). The Wxdialog code is based on the (rather unstable and now quite outdated) Xdialog v1.2.0 code. IMHO Wxdialog is rather pointless, given the features Xdialog provides (including the wizard mode, the multiple inputs boxes, the progress report box, the check button, etc)...
- **whiptail**: a dialog clone which I heard about but that I never tested (IIRC it uses newt instead of ncurses).

Xdialog high compatibility mode:

As a drop in replacement for **dialog**, **cdialog** and **gdialog**, Xdialog tries to maintain the highest degree of compatibility wherever possible. Because of the differences between a ncurses based console utility and a GTK+ based X11 utility, this compatibility is not perfect though. Also, the Xdialog development history introduces its own oddities and some of the default Xdialog behaviours are different from those of (c)dialog...

I came to the conclusion that the only way to keep the highest possible compatility without sacrifying any of the new features offered by Xdialog, was to make it behave in two possible ways, depending on its actual usage in each script (either as a drop in replacement or as a full featured independent utility).

The approach I adopted is to allow Xdialog behaviour customisation using specific environment variables. The advantage of this solution is that while modified with some environment variable assigments, the script is still usable with (c)dialog.

As of v2.0.0, Xdialog checks for the existence of the **XDIALOG_HIGH_DIALOG_COMPAT** environment variable; if this variable is set to "**1**" or "**true**" (case insensitive), then Xdialog behaves in the closest way to (c)dialog (dropping some of its features and adopting (c/g)dialog defaults and syntax), the net effect being that:

- a fixed (monospacing) font is used in all Xdialog GTK+ widgets (including labels, tags, menu/list items and buttons);
- the --left common option is used as the default justification option instead of the --center one;
- the <text> wrapping is always in force (--wrap) and the --fill option is ignored.
- the --no-cr-wrap common option is used as the default instead of the --cr-wrap one (but it does not apply to the <backtitle> in this case).

- regardless of the specified box size, the auto-sizing feature of Xdialog is forced when **XDIALOG_FORCE_AUTOSIZE** environment variable is set to "**1**" or "**true**". This makes for the sizing problems of some menus (e.g. when the specified box size is actually too small for the number of specified menu/list entries to fit, or when a backtitle is specified; with (c)dialog the backtitle is not held into the boxes themselves, unlike Xdialog which needs therefore for more room in its widgets).
- the *Cancel* button is not displayed into the tailbox and textbox widgets;
- the infobox widget is turned into a msgbox unless the **XDIALOG_INFOBOX_TIMEOUT** environment variable is set (in 1/1000s) and greater than 0, in which case an actual infobox **without button** is used;
- the --version special option returns the same string as --print-version.

In order to take benefit of this improved compatibility in an existing (c)dialog script, you just have to add the following line to the script before the first invocation of Xdialog:

set XDIALOG_HIGH_DIALOG_COMPAT=true
or, if you use **bash**:
export XDIALOG_HIGH_DIALOG_COMPAT=true

Then, before each Xdialog invocation in the script, you may customize the behaviour of Xdialog by setting the **XDIALOG_FORCE_AUTOSIZE** and **XDIALOG_INFOBOX_TIMEOUT** environment variables.

Simple, isn't it ?

Other compatibility features:

By default and wether **XDIALOG_HIGH_DIALOG_COMPAT** is set or not, Xdialog does its best to accept (c)dialog syntax and options:

- the cdialog **--passwordbox** box option is accepted by Xdialog and interpreted as --password --inputbox.
- the (c)dialog **--menu** box option is accepted by Xdialog and interpreted as --menubox.
- the cdialog **--defaultno** option is accepted by Xdialog and interpreted as --default-no.
- Because the gauge widget is wrongly spelled as "guage" in some releases of (c/g)dialog (this is what happens when the programmer is dyslexic ;-) Xdialog also accepts the **--guage** option for the gauge widget... Of course as you are not dyslexic (are you ?), you should only use **--gauge** in your own Xdialog scripts !
- Xdialog ignores any unknown transient/common option (and any associated parameter) passed into its command line. You may therefore use Xdialog in place of (c)dialog in an existing script without removing the options that are irrelevant to Xdialog (such as --clear, --no-kill, --and-widget, etc...).
- Xdialog accepts a standalone **--clear** option in its command line (this does nothing but Xdialog does not complain about a missing box option).
- A small wrapper (Xdialog.wrapper) is available and may be installed by running the install-wrapper script from the samples directory, just type:

  cd <path_to_Xdialog_doc_directory>/samples;./install-wrapper

  This wrapper will make any script using (c)dialog to call automatically Xdialog when a X display is available (if no X display is available, then the actual (c)dialog binary is called instead); to ensure the best possible compatibility, Xdialog is called by Xdialog.wrapper in high compatibility mode with forced autosize feature on (unless the **XDIALOG_HIGH_DIALOG_COMPAT** and/or the

XDIALOG_FORCE_AUTOSIZE environment variables are already set, in which case their value is not changed by Xdialog.wrapper).

Pending compatibility issues:

Some Xdialog widgets can't be made 100% (c)dialog compatible:

- The infobox which behaviour in (c)dialog is to print a box on the console and return immediately to the calling script without clearing the console (the "box" therefore staying displayed); when the next menu is setup, (c)dialog overwrites the infobox. This behaviour can't be reproduced with GTK+ menus although this could be coarsely simulated by forking Xdialog when such a menu is needed (but so far I could not figure out how to fork successfully with GTK+...). With Xdialog, you will have to set **XDIALOG_HIGH_DIALOG_COMPAT** to "true" and then to choose which type of infobox to use by setting (or not) **XDIALOG_INFOBOX_TIMEOUT** accordingly: whatever type you choose, the script will be suspended until the infobox is closed (either after the timeout period or when the user presses the *OK* button)...
- Although allowed by Xdialog, the --tailboxbg option will not make Xdialog to fork nor to run in the background: an explicit "&" has to be appended to the Xdialog command line IOT run it as a background process.

Xdialog documentation - FAQ

FAQ contents:

1. Can I use Xdialog from a Perl script ?
2. How to recover the output of Xdialog in the Perl script then ?
3. How to make Xdialog windows to appear at a given position (in absolute coordinates) on the screen ?
4. What are the ways to customize Xdialog windows look for a given script ?
5. How could I make the printer used by Xdialog user-dependant ?
6. Why does the --fill option of Xdialog sometimes fails to make the text fit the boxes ?
7. How to force the use of fixed width fonts in all Xdialog widgets ?
8. How to select hidden files/directories with the file/directory selector ?
9. How to build dynamically a menu using Xdialog ?
10. I just compiled Xdialog but I sometimes get core dumps or segfaults with some menus: what's wrong with it ?

---

1. Can I use Xdialog from a Perl script ?

Yes, in fact you can use Xdialog from any scripting language supporting external command calls.

2.How to recover the output of Xdialog in the Perl script then ?

I found three ways (but there are perhaps more; I'm not a Perl guru !):

a.- Using the "system" command (not the best way IMHO):

Just redirect the Xdialog output into a temp file, then use the "open" command to read the result, put it in a table and use it (here just printing it):

```
system('Xdialog --inputbox "Please enter something below:" 0 0 2>/tmp/my_temp_file');
if ($? == 0) {
        open(RESULT, "/tmp/my_temp_file");
        @result=<RESULT>;
        close(RESULT);
        print @result;
}
```

Finally, destroy the temp file (may be there is a better way to do this...):

```
system("rm -f /tmp/my_temp_file");
```

b.- Using backquotes (just like in "sh": IMHO the best way...):

```
$result=`Xdialog --stdout --inputbox "Please enter something below:" 0 0`;
if ($? == 0) {
        print $result;
}
```

c.- Using "open" and streams (useful if you want the result to be put in a table):

```
open(RESULT, 'Xdialog --stdout --inputbox "Please enter something below:" 0 0 |');
if ($? == 0) {
        @result=<RESULT>;
        print @result;
}
close(RESULT);
```

Note the use of --stdout in (b) and (c) so to send the result returned by Xdialog to the proper output stream...

3.How to make Xdialog windows to appear at a given position (in absolute coordinates) on the screen ?

There are two ways but the result is not guaranteed as some window managers will simply ignore or override the GTK+ placement requirements and place the windows where they feel like.

a.- As of v2.0.0, Xdialog takes into account the origin coordinates given into a "-geometry"-like parameter; e.g. passing 240x120+150+80 as a size parameter to Xdialog will place its window (which size will be 240x120 pixels) at Xorg=150 and Yorg=80. In this example, you may as well let Xdialog auto-size by passing 0x0+150+80 instead. E.g;:

```
Xdialog --title ppp.log --tailbox /var/log/ppp.log 0x0+150+80
```

b.- Some window manager do allow to place a window with a given title or class name at a given position on the screen. Xdialog therefore provides a way to set its window manager class name through the --wmclass option. E.g.:

```
Xdialog --wmclass ppp_log_tailbox --title ppp.log --auto-placement \
      --tailbox /var/log/ppp.log 0 0
```

Now this Xdialog tailbox is registered with the "ppp_log_tailbox" name. With twm and fvwm(2/95) you will have to edit the .Xdefaults file in your home directory, adding "ppp_log_tailbox*geometry" parameters so to set the Xdialog position and/or size. With sawfish, just move the Xdialog window to the place of your choice, pull down the window manager options menu (clicking on a given button in the Xdialog window title bar or on the title bar itself: this is user configurable and may also depend from your sawfish theme) and choose the "History"/"Remember position" item in the menu; the next time an Xdialog window with a "ppp_log_tailbox" wmclass will be open, it will pop up at the remembered position...

4.What are the ways to customize Xdialog windows look for a given script ?

a.- Windows decorations:

Through --wmclass option provided your window manager makes use of the wmclass name of the windows so to decorate them differently. The method is the same as in §3 (just use the Xdialog --wmclass option and RTFM of your window manager; hints: "winoptions" editing for IceWM, window manager option menu use for sawfish, ".Xdefaults" editing for twm/fvwm, etc...).

b.- GTK+ themes:

As of v1.4.6, Xdialog accepts a new --rc-file option. Thanks to this feature Xdialog can be instructed to use a given GTK+ theme (which may therefore be different from the theme currently in use).

c.- User defined icons:

As of v1.4.6, Xdialog accepts the new --icon option that will make a user defined icon (in XPM format only) to appear on the left of the <text> (for the Xdialog box options using this parameter, the box options without a <text> parameter in their syntax are not taking the --icon option into account).

5.How could I make the printer used by Xdialog user-dependant ?

The name of the printer to be used by Xdialog is to be passed after the --print option in the Xdialog command line. Nothing prevents you to make this printer name a variable which will be set via an sh include file. Here is an example of how to do it:

```
#!/bin/sh
```

```
# Sample script with per-user costumizable printer name.

# First set the default printer.
XDIALOG_PRINTER="lp"

# Check if the user wants to use its own printer.
if [ -f ~/.xdialog-printer] ; then
   . ~/.xdialog-printer
fi

Xdialog --print $XDIALOG_PRINTER .../...
```

Then for each user, the following **.xdialog-printer** file may be put in its home directory:

```
# /home/foo/.xdialog-printer include file for user "foo".

# Let's use the "bar" printer...
XDIALOG_PRINTER="bar"
```

6.Why does the --fill option of Xdialog sometimes fails to make the text fit the boxes ?

Because this option must use the GTK+ auto-wrapping mode and alas this only really works when the box auto-sizing feature of GTK+ is used. The work around is therefore to let GTK+ calculate the size of the box by passing a 0x0 (or 0 0) size to Xdialog.

7.How to force the use of fixed width fonts in all Xdialog widgets ?

With some scripts written for **dialog**, some pre-formatted text may appear mis-aligned in Xdialog menus. This is because GTK+ uses proportional fonts while the console tools such as dialog may only use fixed width fonts.

While the --fixed-font intructs Xdialog to use a fixed width font (more exactly a monospacing one) into the text windows of the tailbox, textbox and editbox widgets, the labels and backtitle of Xdialog still use the font defined in the GTK+ theme in force when Xdialog is started. There are two ways around this, by using either the (c)dialog high compatibility mode, or the --rc-file option together with a gtkrc file where the font parameter is set for a monospacing font name. E.g.:

```
style 'fixed_font' {
 font = "-*-*-medium-r-normal-*-*-*-*-*-m-70-iso8859-1"
}
widget '*' style 'fixed_font'
```

8.How to select hidden files/directories with the file/directory selector ?

In the exact same way as with any software using the GTK+ file selector: type a "." into the text entry field and then press the TAB key: the hidden files/directories will then be presented into the selector lists, allowing you to select one with the mouse.

Also, appending ".*" to the default directory name into the Xdialog command line, makes all the hidden file/directory names (and only them) to appear into the fselect/dselect widgets when poped up. E.g.:

Xdialog --fselect "/home/foo/.*" 0 0

9.How to build dynamically a menu using Xdialog ?

You may need to build dynamically a menu for some scripts. Although Xdialog does not accept to take its parameters from a file (in which you could put the menu entries), nothing prevents you to build dynamically a sub-script containing a Xdialog command, and then call it from you main script. Take a look to the xlock-wrapper script which uses such a trick.

10. I just compiled Xdialog but I sometimes get core dumps or segfaults with some menus: what's wrong with it ?

There is currently no known bug in Xdialog code (and I use Xdialog on four different Linux systems without problem so far). But here are some hints on why it may fail to run on some systems:

a.- Xdialog can **theorically** be used with GTK+ v1.2.0 and upper but it has only been extensively tested with GTK+ v1.2.8 to v1.2.10: if you are using an older GTK+ version, please upgrade (some segfaults have been reported when using GTK+ v1.2.6 that disappeared once upgraded to GTK+ v1.2.8).

b.- GCC is a fine compiler, but GCC v2.95 is broken ! It sometimes fails to notice that the stack has been tidied on return of self-tidying functions (mostly math functions)... As a result, compiling any program with the **-fomit-frame-pointer** flag may result in instable binaries (or even instable Linux kernel: I recompiled mine with **-fno-omit-frame-pointer** and some strange crashes I got in the past are now history !): If you compile Xdialog with this flag you **WILL** get into trouble... Please use the **-fno-omit-frame-pointer** flag when compiling Xdialog with gcc v2.95.x (as of Xdialog v1.5.0, the **configure** script takes care of adding automatically this compile option when a bugged gcc version is detected) !

The x86 binary RPMs on http://xdialog.free.fr are compiled (on a Mandrake v7.2 distro) with the proper settings and dynamically linked to glibc v2.1.3, XFree86 v4.1.0 and GTK+ v1.2.10.

If you meet the requirements above and still get segfaults with Xdialog, then this is probably a Xdialog bug that you should report to me; please be precise and give example(s) of how to reproduce the bug (I am generally pretty quick to diagnoze/fix bugs provided they were properly reported).