

# Testing C++ Qt GUI applications

Rich l Bilderbeek

May 6, 2016

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Qt . . . . .	1
1.2	xdotool . . . . .	1
1.3	Travis CI . . . . .	2
<b>2</b>	<b>Setting up a minimal project</b>	<b>2</b>
2.1	Creating a minimal Qt application . . . . .	2
2.2	Testing this minimal Qt application . . . . .	6
<b>3</b>	<b>Enter secret code</b>	<b>7</b>
3.1	Creating the application . . . . .	8
3.2	Testing the application . . . . .	8
<b>A</b>	<b>xdotool minimal tutorial</b>	<b>10</b>

## 1 Introduction

This is ‘Testing C++ Qt GUI applications’, version 0.1, a tutorial about testing C++ GUI applications written with Qt.

For now, I only consider Qt applications under GNU/Linux, as I use ‘xdotool’ to manipulate windows.

Goal is to reliably test Qt GUI applications.

### 1.1 Qt

Qt is a cross-platform C++ library to create GUIs.

### 1.2 xdotool

xdotool is a GNU/Linux command-line tool.

## 1.3 Travis CI

Travis CI is a continuous integration (hence the 'CI') tool.

## 2 Setting up a minimal project

Setting up a minimal project consists out of these steps:

- Creating a minimal Qt application
- Testing the minimal Qt application

### 2.1 Creating a minimal Qt application

Create a new Qt Creator project, by clicking from the Qt Creator menu 'File | New File or Project'.

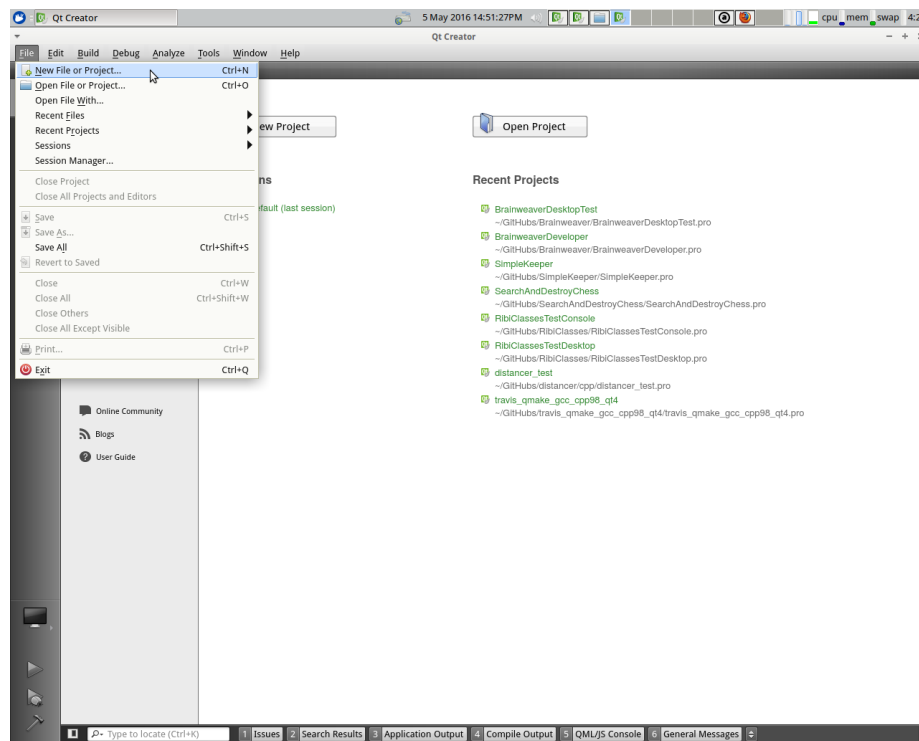


Figure 1: Create a new project

In the 'New' dialog, select 'Qt Widgets Application' and click 'Choose'.

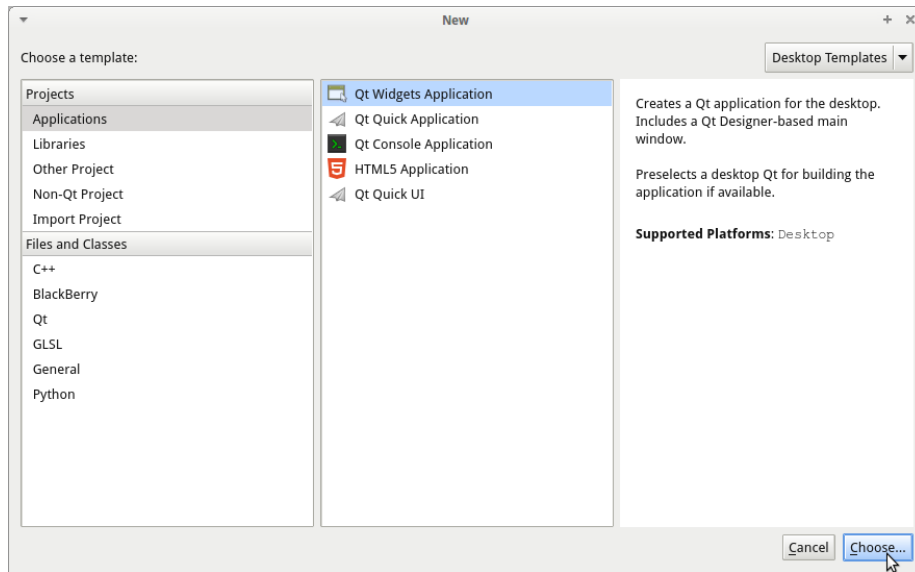


Figure 2: Create a new Qt Widgets application

In the 'Qt Widgets Application' dialog, pick a suitable location to put the files of your project.

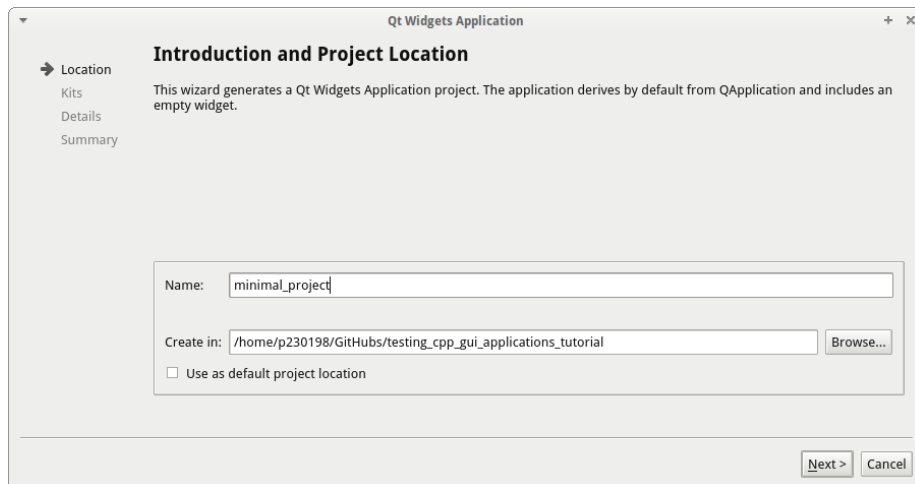


Figure 3: Give your new application a suitable location

In the next 'Qt Widgets Application' dialog, just use the default kits by clicking 'Next'.

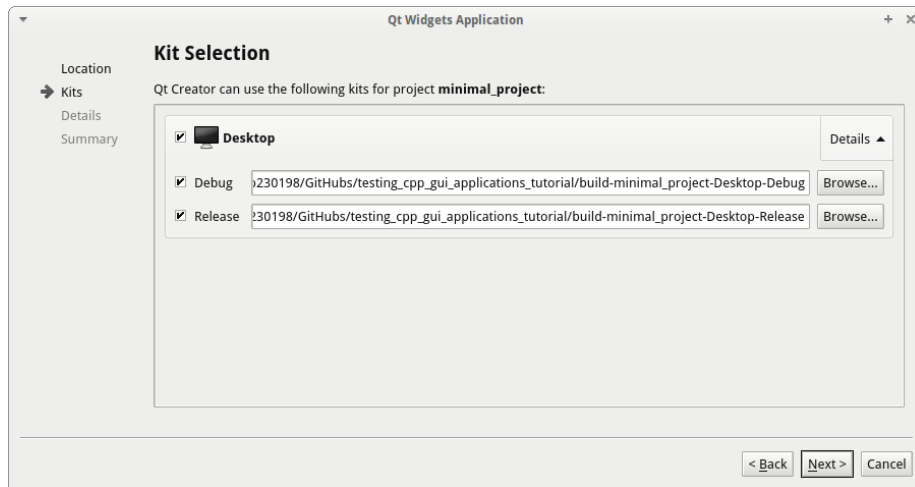


Figure 4: Use the default kits

In the next 'Qt Widgets Application' dialog, use 'QDialog' as the base class. Use all default names by clicking 'Next'.

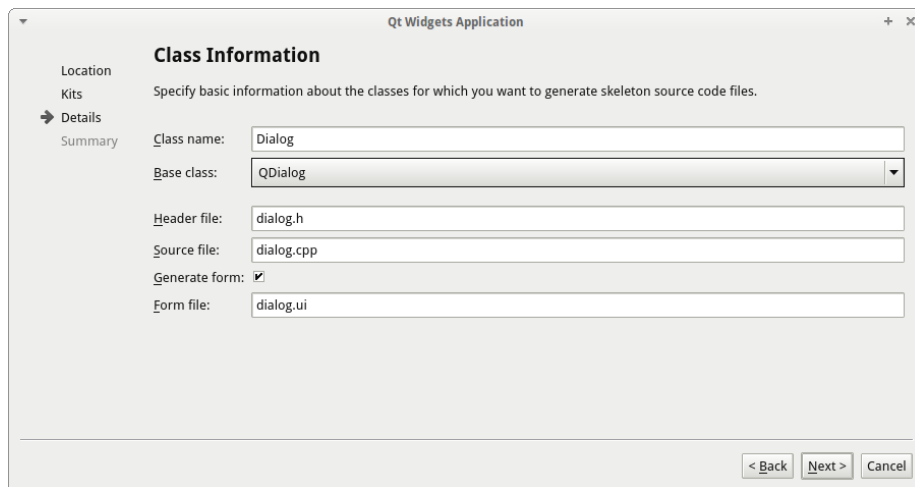


Figure 5: Set the base class of your application's main window to QDialog

In the next 'Qt Widgets Application' dialog, skip this way of using git to version control by clicking 'Finish'.

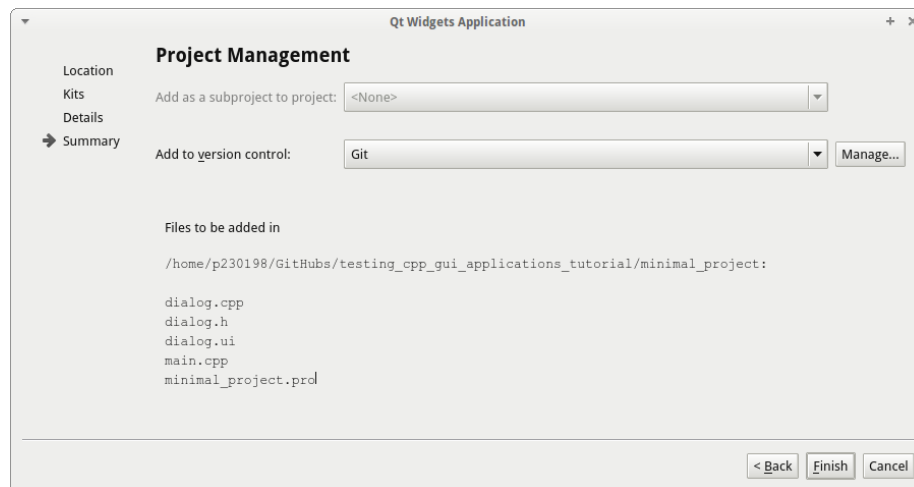


Figure 6: Skip using git via Qt Creator

Now, your minimal project is created.

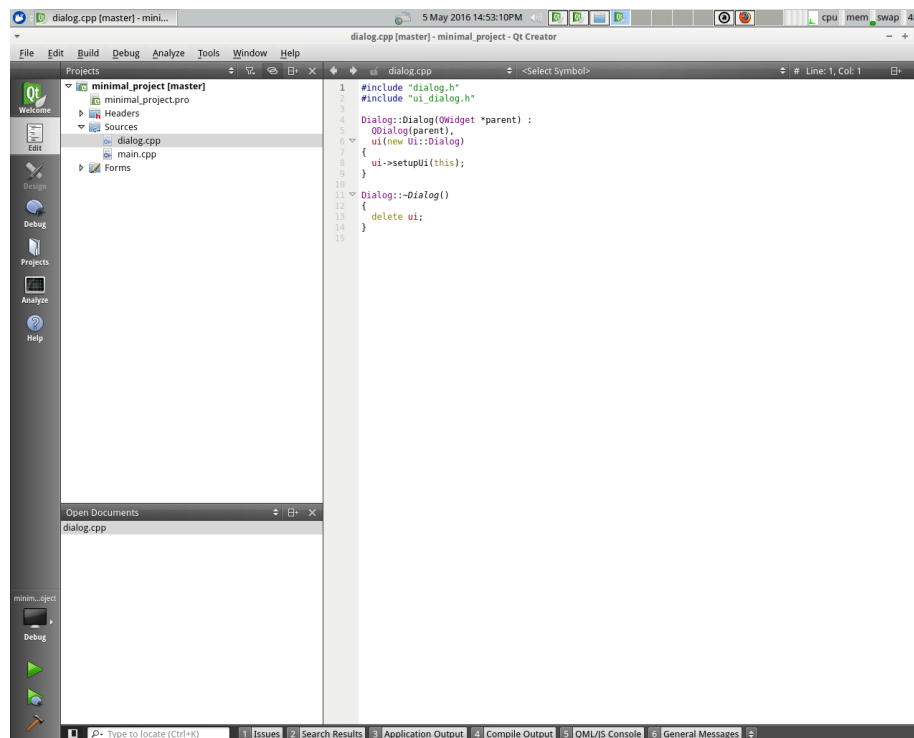


Figure 7: Part of the code of your first minimal project

Now run the application, by either pressing `CTRL+R`, selecting 'Build | Run' from the Qt Creator menu, or click the green arrow on the left.

The minimal application is an empty dialog.

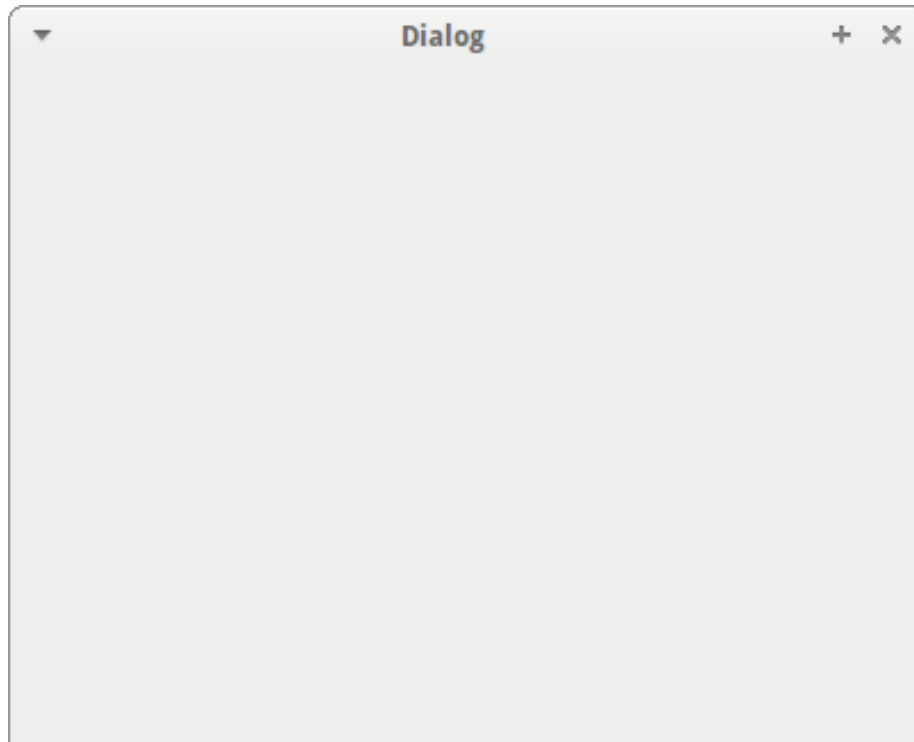


Figure 8: The minimal application

## 2.2 Testing this minimal Qt application

Instead of using the low-level `xdotool` commands as described in chapter A, the minimal Qt application bash testing script is tested with higher-level functions:

---

**Algorithm 1** minimal\_project.sh

---

```
#!/bin/bash
myexe="minimal_project"
dialog_name="Dialog"

echo "Starting_the_application"
./$myexe &
sleep 1

echo "Check_if_the_dialog_can_be_found"
. ../scripts/get_dialog_id.sh
id='get_dialog_id $dialog_name'

echo "Close_the_dialog_using_ALT-F4"
. ../scripts/close_first_dialog_with_name.sh
close_first_dialog_with_name $dialog_name

echo "Starting_the_application_again"
./$myexe &
sleep 1

echo "Close_the_dialog_by_setting_the_mouse_at_the_
      closing_glyph_and_clicking"
. ../scripts/
  set_mouse_at_close_glyph_of_first_dialog_with_name.sh
set_mouse_at_close_glyph_of_first_dialog_with_name
  $dialog_name
sleep 1 # For suspense
# Set the mouse at the closing glyph again (in case the
# user moved the mouse)
set_mouse_at_close_glyph_of_first_dialog_with_name
  $dialog_name
xdotool click 1
```

---

It opens the application, closes it by using ALT-F4, re-starts the application, then closes it by using the mouse.

Note that this scripts is stripped down to fit on one page.

### 3 Enter secret code

In this project, we create an application in which a secret code must be entered

- Creating the application

- Testing the application application

### 3.1 Creating the application

Here it is:

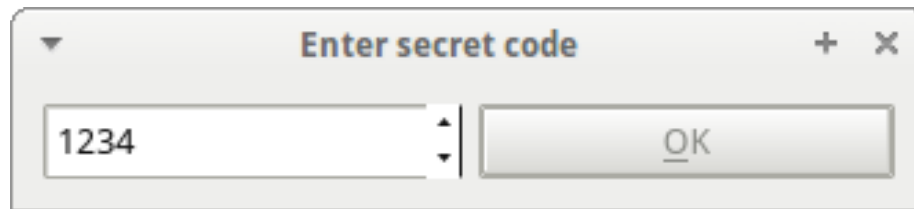


Figure 9: The application

### 3.2 Testing the application

Here code:



---

**Algorithm 2** enter\_secret\_code.sh

---

```
#!/bin/bash
myexe="enter_secret_code"
dialog_name="Enter_secret_code"

echo "Starting_the_application"
./$myexe &
sleep 1

echo "Check_if_the_dialog_can_be_found"
. ../scripts/get_dialog_id.sh
id='get_dialog_id $dialog_name'

echo "Set_the_value_1111_to_test"
xdotool windowactivate $id key Delete key Delete key
Delete key Delete type 1111

echo "Click_the_OK_button_fails"
xdotool windowactivate $id sleep 0.1 key Tab sleep 0.1
key Return
if [ ! 'get_dialog_id $dialog_name' ]
then
    echo "Error:_should_not_be_able_to_close_dialog_with_
    incorrect_code"
    exit 1
fi

echo "Set_the_value_4242_to_test"
xdotool windowactivate $id key BackSpace key BackSpace
key BackSpace key BackSpace type 4242 sleep 0.1 key
alt-o

echo "Click_the_OK_button"
xdotool windowactivate $id sleep 0.1 key Tab sleep 0.1
key Return
if [ 'get_dialog_id $dialog_name' ]
then
    echo "Error:_should_not_have_the_dialog_open_anymore"
    exit 1
fi
```

---

Again this is a stripped-down version

## A xdotool minimal tutorial

This is a step-by-step minimal xdotool tutorial.

**Starting the minimal Qt application** Start the minimal Qt application in chapter 2.1. The minimal application will start (as shown in figure 8) with window name 'Dialog'. To start this application from the command line:

```
./minimal_example &
```

Note the ampersand ('&') after the execution call of 'minimal\_example'. It will launch the example on a new thread.

**xdotool cannot detect a Qt application window from its title** 'xdotool' can theoretically be used to detect that window with 'Dialog' as its title with the following command:

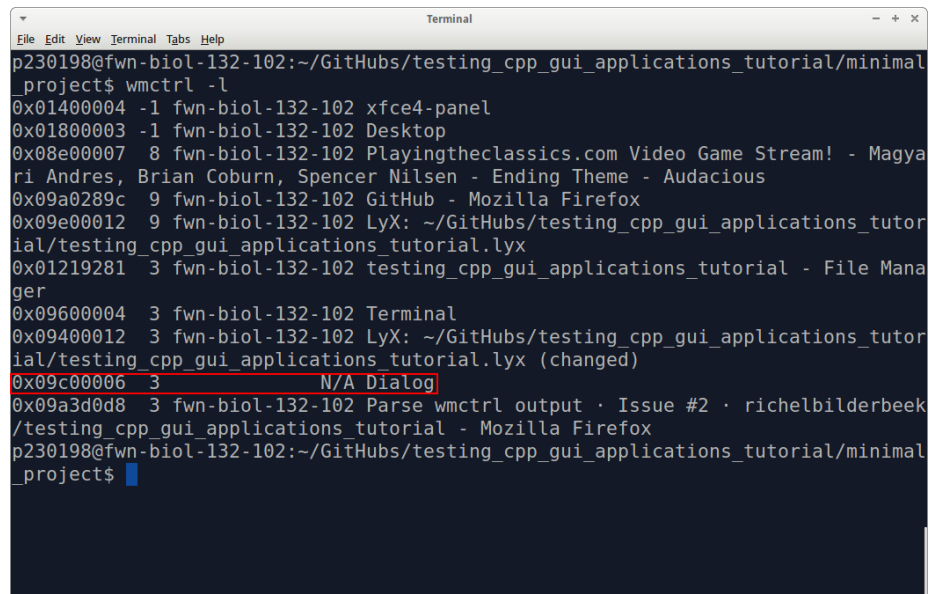
```
xdotool search --name "Dialog"
```

Too bad, this does not return any hit. We will need a detour to be able to find our window.

**wmctrl can find all window title** The program 'wmctrl' can give us all windows:

```
wmctrl -l
```

Here you can see that wmctrl can detect our window:



```
p230198@fwn-biol-132-102:~/GitHubs/testing_cpp_gui_applications_tutorial/minimal_project$ wmctrl -l
0x01400004 -1 fwn-biol-132-102 xfce4-panel
0x01800003 -1 fwn-biol-132-102 Desktop
0x08e00007 8 fwn-biol-132-102 Playingtheclassics.com Video Game Stream! - Magya
ri Andres, Brian Coburn, Spencer Nilsen - Ending Theme - Audacious
0x09a0289c 9 fwn-biol-132-102 GitHub - Mozilla Firefox
0x09e00012 9 fwn-biol-132-102 LyX: ~/GitHubs/testing_cpp_gui_applications_tutor
ial/testing_cpp_gui_applications_tutorial.lyx
0x01219281 3 fwn-biol-132-102 testing_cpp_gui_applications_tutorial - File Mana
ger
0x09600004 3 fwn-biol-132-102 Terminal
0x09400012 3 fwn-biol-132-102 LyX: ~/GitHubs/testing_cpp_gui_applications_tutor
ial/testing_cpp_gui_applications_tutorial.lyx (changed)
0x09c00006 3 N/A Dialog
0x09a3d0d8 3 fwn-biol-132-102 Parse wmctrl output · Issue #2 · richelbilderbeek
/testing_cpp_gui_applications_tutorial - Mozilla Firefox
p230198@fwn-biol-132-102:~/GitHubs/testing_cpp_gui_applications_tutorial/minimal_project$
```

Figure 10: wmctrl finds a window with Dialog as its title

xdotool can work from a window ID, which is the first number wmctrl shows (in this case the number is '0x09c00006' ('0x' denotes the number is hexadecimal)). Next step is to extract the window ID from this output.

**Extract the window ID from wmctrl its output** To extract the window ID from wmctrl its output, we will select the line from wmctrl that contains the text 'Dialog' using egrep:

```
wmctrl -l | egrep "Dialog"
```

Note the pipe ('|') symbol, which is a UNIX symbol to use the first command its output as input for the second. In this case, this will return:

```
0x09c00006 3 N/A Dialog
```

We can select the first field using the 'cut' command:

```
wmctrl -l | egrep "Dialog" | cut -f 1 -d ' '
```

Here the single line is cut into fields by using spaces as a delimiter, where we select the first field. This results in the hexadecimal value of the window ID:

```
0x09c00006
```

**Use the extracted window ID to activate a window with xdotool** Because xdotool can handle hexadecimal, we can activate our dialog with this command:

```
xdotool windowactivate $(wmctrl -l | egrep "Dialog" | cut -f 1 -d ' ')
```

From this, we can do all kinds of tests.

**xdotool: close a window by using ALT+F4** xdotools chains command, so now you can add the more useful commands at the end of the line.

For example, we can close the minimal dialog like this:

```
xdotool windowactivate $(wmctrl -l | egrep "Dialog" | cut -f 1 -d ' ') sleep 0.1
```

Note that I added a call to 'sleep', to give the window some time to be activated.

**xdotool: get a window its geometry** Or get the window its geometry:

```
xdotool getwindowgeometry $(wmctrl -l | egrep "Dialog" | cut -f 1 -d ' ')
```

This results (on my computer) in:

```
Window 48234502
Position: 441,386 (screen: 0)
Geometry: 400x300
```

**xdotool: move the mouse cursor to the window close glyph** From that I can conclude where I have to put my mouse cursor to hover above the closing cross of the dialog:

```
xdotool windowactivate $(wmctrl -l | egrep "Dialog" | cut -f 1 -d ' ') sleep 0.1
```

**xdotool: close a window by clicking on the window close glyph** Adding a click command after this, will cause the window to close:

```
xdotool windowactivate $(wmctrl -l | egrep "Dialog" | cut -f 1 -d ' ') sleep 0.1
```

With these snippets of knowledge, one can create a decent test script

## References