Tutorial for wxWindows: a portable GUI toolkit for C++

Julian Smart
Artificial Intelligence Applications Institute
University of Edinburgh
EH1 1HN

January 1993

1 Introduction

This short tutorial accompanies the main wxWindows 1.30 manual, and takes a look at some of the supplied demonstration programs. The tutorial is incomplete and may be expanded in later releases.

2 The demo programs

2.1 A minimal wxWindows program

The best way to get a feel for how to use a tool is to see a small example. The supplied demo 'minimal' (source file `minimal.cc`) shows a rudimentary wxWindows program. It has a main window with a panel inside it, displaying a message. There is a menu bar with a **File** menu which in turn has a **Quit** option, and the program has its own icon. Under Windows 3, the system menu shows the usual options including Minimize, Maximize and Close, and under X, there is a similar pull-down system menu provided by the current window manager. The window is resizeable, and the panel automatically resizes to fit its parent.

Look at `minimal.cc`.

```
/*
 * File:     minimal.cc
 * Purpose:  Minimal wxWindows app
 *
 *                        wxWindows 1.40
 * Copyright (c) 1993 Artificial Intelligence Applications Institute,
 *                    The University of Edinburgh
 *
 *                      Author: Julian Smart
 *                         Date: 18-4-93
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose is hereby granted without fee, provided
```

```
 * that the above copyright notice, author statement and this permission
 * notice appear in all copies of this software and related documentation.
 *
 * THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS,
 * IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
 * MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
 *
 * IN NO EVENT SHALL THE ARTIFICIAL INTELLIGENCE APPLICATIONS INSTITUTE OR THE
 * UNIVERSITY OF EDINBURGH BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR
 * CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM
 * LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF
 * DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH
 * THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

#include <windows.h> // Included only for benefit of MSC7 precompiled headers
#include "wx.h"

// Define a new application type
class MyApp: public wxApp
{ public:
    wxFrame *OnInit(void);
};

// Define a new frame type
class MyFrame: public wxFrame
{ public:
    MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h);
    void OnMenuCommand(int id);
};

// ID for the menu quit command
#define MINIMAL_QUIT 1

// This statement initializes the whole application and calls OnInit
MyApp myApp;

// 'Main program' equivalent, creating windows and returning main app frame
wxFrame *MyApp::OnInit(void)
{
  // Create the main frame window
  MyFrame *frame = new MyFrame(NULL, "Minimal wxWindows App", 50, 50, 400, 300);

  // Give it an icon
  frame->SetIcon(new wxIcon("aiai_icn"));

  // Make a menubar
  wxMenu *file_menu = new wxMenu;
  file_menu->Append(MINIMAL_QUIT, "Quit");
  wxMenuBar *menu_bar = new wxMenuBar;
  menu_bar->Append(file_menu, "File");
  frame->SetMenuBar(menu_bar);

  // Make a panel with a message
```

```
  wxPanel *panel = new wxPanel(frame, 0, 0, 400, 300);
  (void)new wxMessage(panel, "Hello, this is a minimal wxWindows program!", 0, 0);

  // Show the frame
  frame->Show(TRUE);

  // Return the main frame window
  return frame;
}

// My frame constructor
MyFrame::MyFrame(wxFrame *frame, char *title, int x, int y, int w, int h):
  wxFrame(frame, title, x, y, w, h)
{}

// Intercept menu commands
void MyFrame::OnMenuCommand(int id)
{
  switch (id) {
    case MINIMAL_QUIT:
      delete this;
    break;
  }
}
```

The statement `#include "wx.h"` provides the program with access to all the wxWindows classes and functions.

The first class declaration, **MyApp**, declares a new application, overriding one member function **OnInit**. This is an essential part of writing a wxWindows program, since **OnInit** is the equivalent of **main** in a normal C++ program.

The class **MyFrame** declares a constructor, and a message handler for intercepting menu commands.

The definition of the global variable **myApp** looks innocuous enough but this starts the whole application going simply by being defined.

The **OnInit** MyApp member function does the initialization of the program. It creates a main frame, sets the icon, creates a menu bar, a panel, and a panel item.

The **MyFrame** constructor may seem a little pointless, but it fulfils the requirements of Microsoft C/C++ syntax in defining the constructor in terms of its parent's constructor.

The **OnMenuCommand** definition intercepts menu commands for the main frame. If the option chosen is **Quit**, the application terminates by deleting the main frame. Normally any other existing frames should be deleted (subframes are deleted automatically); these calls are usually put in the frame's **OnClose** handler so that a system-generated **OnClose** event will enable the program to clean itself up first. System-generated **OnClose** events delete the main frame after calling **OnClose**, so this should not be done from within **OnClose**.

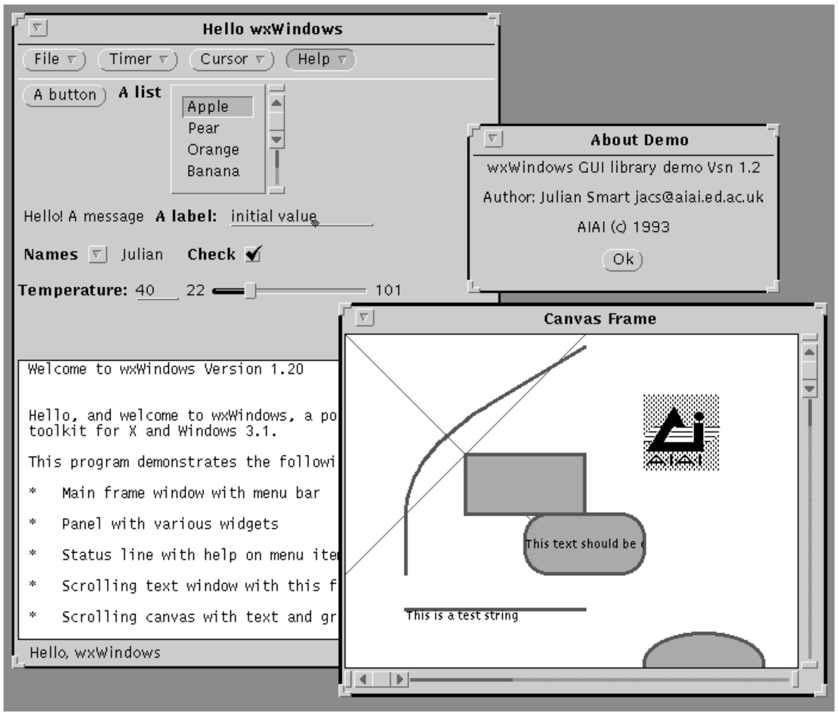

Figure 1: Demo program running under X

2.2 More advanced features: the *hello* demo

The 'hello' (source files, `hello.cc` and `hello.h` shows off some more wxWindows features (see Figures 1 and 2). When run, two windows pop up. One is the 'main window', with two subwindows - a panel containing various 'widgets', and a text window. The other contains a canvas, drawing some simple shapes, and allowing the user to doodle on it by dragging with the left mouse button. The canvas contents can be scaled and printed out, either to a printer supported by Windows or to PostScript, writing to a file or invoking the printer directly. Under Windows, the graphic may be copied to the clipboard as a metafile.

Both frames can be resized, and the subwindows will be resized in an appropriate manner. The text subwindow can be scrolled; on the panel, a button can be pressed for the program to prompt the user with text with which to set the status bar. Clicking on the list box writes a line of text into the text window.

The **File** menu has options for selecting the 'mapping mode' (logical dimensions) used in drawing graphics, a zoom option, and an option for loading a file into the text subwindow using a file selector tool.

The **Timer** menu allows the user to switch a timer on and off; when on, some text gets written to the text subwindow every five seconds.

The **Cursor** menu enables the canvas cursor to be changed, and lets the potential wxWindows programmer view the available standard cursors.

The **About** option of the Help menu pops up a dialog box with some information.

This represents a fair amount of GUI functionality for a relatively small program. This is

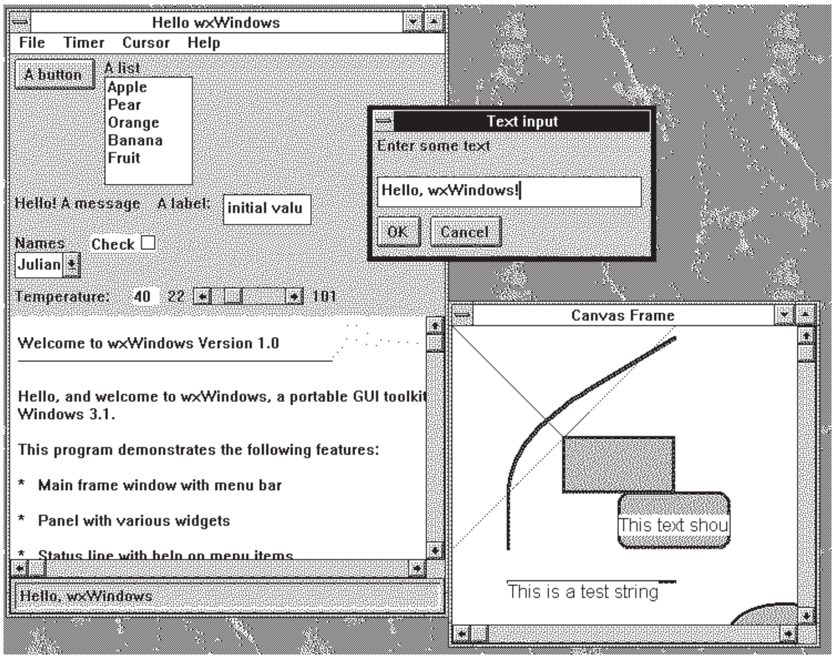

Figure 2: Demo program running under Windows 3

because wxWindows calls are high level (creating a working text window is a single call) and because an object-oriented approach is taken, where much default functionality is provided.

Note also the lack of explicit coordinates or sizes in the panel item creation calls. This is the preferred approach, leaving wxWindows to lay out the items from left to right and top to bottom, with the user interjecting the occasional **NewLine** call. Explicit positioning is not recommended since it is less device independent, but can be achieved by using more parameters to the creation calls, or by using **SetSize** after an item has been created. Coordinates and sizes default to -1, which tells wxWindows to choose appropriate positioning and sizing. In this example, the windows are explicitly sized, but you may size a frame or panel to fit around its contents by calling **Fit**.

MyFrame's OnSize member sizes each subwindow in proportion to the new size of the frame. **MyCanvas's OnPaint** draws a couple of lines, a rectangle and a spline whenever the canvas requires repainting (e.g. on creation, and when exposed). The **OnEvent** member checks for mouse dragging, and draws a line from the last point to the current position. Scrolling the canvas and subsequent repainting is handled automatically by wxWindows.

Finally, two callback functions demonstrate popping up dialog boxes, setting the status line, and inserting text into a text window.

This demo can provide a template for your own application. Gradually modify it for your own needs, and you will rapidly be writing portable X and Windows 3 programs!

2.3 The MDI demo

As explained in the manual, wxWindows takes an automated view of Microsoft Windows's MDI (Multiple Document Interface) since it is a very platform-specific feature. The special MDI 'Window' menu, allowing the user to switch between child windows, is provided automatically, though without any accelerators and without a choice of position. Also, instead of deriving frames from distinct classes for MDI versus SDI, the approach taken in Microsoft's class library, wxWindows uses an option in the frame constructor to switch between styles. This allows the programmer to delay the MDI/SDI decision, perhaps even providing a command-line switch to let the user decide.

The **mdi** example program shows this run-time switching. Invoked without a command line switch, it defaults to SDI. Invoked with the switch `-mdi` it runs as an MDI program, but only under Windows 3.

There are a few extra considerations when programming an MDI applications. One is the choice of menu items. An SDI program might have a main window and several child windows, where the main window menu has options for quitting the program and other global matters, while child window menus have child-specific options. In MDI, the child window menu visually replaces the main window menu when activated, and so it must duplicate some main window menu options. One solution is to include logic to add extra menu items depending on whether MDI or SDI is specified, as the **mdi** example does.

Also, for programs which must be both SDI and MDI (on non-Windows 3 platforms SDI mode is mandatory), the main window must not have subwindows (i.e. panels, canvases or text subwindows) since the client area may be occupied with child MDI windows under Windows 3.

MDI icons pose a small technical difficulty, in that for some reason it is not possible to use the same technique of dynamically painting the icon onto the iconized frame client area as with SDI frames. Consequently the programmer must insert some lines into the resource file for providing icons (see mdi.rc) as well as using the SetIcon call in the program for non-Windows 3 platforms. Only one icon image for child MDI frames may be provided.

2.4 The IPC demo

The demo in the `directory samples/ipc` shows how processes may easily talks to each other synchronously (i.e. when A sends a message to B, A waits for an answer). If you start `server`, then `client`, a new window should appear on top of the server window, which represents the connection between server and client. Quitting the client causes the connection to be broken and this window to disappear.

To illustrate 'hot linking', click on the server's listbox. This sends an *advise* message to the client, telling it to update its own listbox. The reverse is not true, however.

A client may request information from the server. Select the **Request** menu item from the client's **File** menu. The window which pops up is created by the client and contains a message that the server sent back.

Selecting **Execute** from the client's menu makes the server pop up a window. Normally this would execute some command that the client wishes the server to run.

The **Poke** menu item sends a *poke* message to the server; normally this would insert some data into the server's memory.

Interprocess communication in wxWindows uses a subset of DDE (Dynamic Data Exchange) which is Microsoft's standard for low-level IPC under Windows. wxWindows gives you DDE under UNIX as well as Windows, and makes it easier to program into the bargain by using an intuitive object-oriented model of communication.

2.5 The Object Graphics demo

New to version 1.20 of wxWindows is a library of graphic calls to help with manipulating objects on a canvas. The demo in `samples/objects` shows a simple application where the user can create new rectangles and ellipses, join them up, label them, move them about, resize them, and delete them.

This library is not yet documented but I hope that most of it is reasonably intuitive from the header file `graphics.h` and from the demo source code. Some of the features may be tricky to spot, however, such as multiple-segment lines and splines, attachment points on shapes, auto-straightening of line segments, manual repositioning of line endpoints, newlines in text labels (%n).