User Manual for wxWindows Dialog Editor Version 1.7

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

March 1997

Contents

Copyright notice

1. Introduction

The wxWindows Dialog Editor is a tool for creating dialog resource files, in `.wxr` format. It differs from wxBuilder in the following respectes:

1. Scope. It is written for dialog editing only, and is therefore more convenient than wxBuilder for this purpose.
2. File format. Dialog editor reads and writes wxWindows resource files (extension `.wxr`) and has no independent file format.
3. Robustness. It is written in a more principled way than wxBuilder, and is less ambitious.
4. Ease of use. Windows are edited using the mouse or via consistent *property editors*, which provide immediate visual feedback of changed properties.

Dialog Editor should be compiled and used with wxWindows 1.65 or later.

1.1. Current status

Dialog Editor currently runs under MS Windows. It has yet to be tested under Motif and XView: see *Technical notes* (page 8) for a discussion of what needs to be done.

1.2. Future developments

- XView and Motif versions.
- It would be nice to have a dialog browser, showing thumbnails of all dialogs in a particular directory.
- Maybe add a menubar editor (from wxBuilder).
- Maybe convert Windows .rc files.

2. Commands

2.1. Dialog editor menu bar

2.1.1. File menu

New project	Creates a new project (clears index and resets project name).
New dialog	Creates a new dialog resource.
Open	Opens an existing resource file.
Save	Saves the current resources.
Save as	Saves the current resources in a named file.
Clear	Clears the current resources.
Exit	Exit the program.

2.1.2. Edit menu

Recreate	Recreates the currently selected panel item from the underlying resource. This may be necessary to regenerate items that cannot be changed dynamically, and which have got out of sync with the displayed item.
Delete	Deletes the currently selected resource.
Toggle edit/test mode	Toggles from edit to test mode, and vice versa. Edit mode is used for editing dialogs, test mode allows panel items to be manipulated as they will appear to the user.

2.1.3. Help menu

Help topics	Displays on-line help at the contents page.
About	Displays an dialog showing the Dialog Editor version and author.

2.2. Command toolbar

The command toolbar consists of the following tools:


Clears the project.


Opens an existing resource file.


Saves the current resources.


Aligns the centre of the selected items horizontally.


Aligns the top sides of the selected items horizontally.


Aligns the bottom sides of the selected items horizontally.


Aligns the centre of the selected items vertically.


Aligns the left sides of the selected items vertically.


Aligns the right sides of the selected items vertically.

Copies the size of the first selected item to the subsequently selected item(s).


Puts the selected item(s) to the front of the display list.

Puts the selected item(s) to the back of the display list.


Invokes Dialog Editor help.

2.3. Tool palette

The tool palette is used to select a type of panel item to create on the dialog. To create a new panel item, select a tool with left-click, then left-click on the dialog to window. Select the pointer tool to use left-click for selecting and deselecting items.

The tool palette consists of the following tools.

Click this to select/deselect items on a dialog.

Text message item.

Bitmap message item.

Text button item.

Bitmap button item.

Checkbox item.

Radiobox item.

Listbox item.

Choice item.

Single-line text item.

Multi-line text item.

Slider item.

Groupbox item.

Gauge item.

Scrollbar item.

2.4. Resource list

The resource list shows a list of the dialogs, panel items and bitmaps currently loaded in Dialog Editor. Double-clicking on a dialog item shows the associated dialog box.

3. Procedures

3.1. Running Dialog Editor

To run Dialog Editor under Windows, click on the Program Manager or Explorer icon. Under UNIX, run from the command line.

The main window shows a menu bar, command toolbar, tool palette, resource list, and status line.

3.2. Creating a dialog

To create a new dialog, click on the **File: New** menu item, or equivalent toolbar button. A dialog will appear. To put a panel item on the dialog, left-click on the appropriate palette icon and then left-click on the dialog. A new item will appear at the place you clicked.

You can edit any panel item or dialog by control-left clicking. A property editor will appear, allowing any property to be selected and edited (see *Using property editors* (page 4)). You can also edit items by right-clicking to show a menu, and then selecting *Edit properties*.

To move a panel item, drag the item with the left mouse button, or edit the position values in the property editor. To resize a panel item, you can either select it by left-clicking and then dragging on a selection handle, or edit the size values in the property editor.

You can delete items from the right-click menu, or by selecting the item and choosing **Edit: Delete** from the menu bar.

3.3. Using property editors

Property editors consist of a list of properties and current values, plus controls at the top of the editor. If the property is of an appropriate type, you can edit the value directly in the text field, and confirm or cancel the value using the two buttons to the left of it. If the property has a predefined range of values, such as labelFontFamily, you can see a list of permissable values by clicking on the button labelled with an ellipsis symbol (**...**). This will show a listbox with possible values and current selection. You may also be able to cycle through values by double-clicking the value in the listbox.

Properties may have special editors appropriate to the type. Filename properties invoke the file selector, and properties containing list of user-definable strings use a string editor.

When you change a property value, this value is immediately reflected in the dialog or panel item. If the item allows this value to be changed dynamically, the relevant wxWindows function will be called internally to effect the change. If the value cannot be changed dynamically, the item will be destroyed and re-created, which means that there will be more flickering associated with some kinds of property changes than others.

3.4. Saving and loading files

Use *File: Save* and *File: Save as* or the equivalent toolbar button to save the current dialog(s) in a wxWindows resource file (extension `.wxr`).

The `.wxr` file can be used directly in a wxWindows program, if wxWindows resources have been enabled when building the wxWindows library. These files can be loaded dynamically, or included directly into program source with a `#include` directive. See the wxWindows user manual for further details.

3.5. Multi-platform development

`.wxr` files generated on one environment (e.g. Windows) can be used in another (e.g. Motif). However, because the same panel item can have different sizes on different GUIs, the user should be cautious in assuming that one resource file will work for all platforms. It may be better to plan to conditionally include or load different resource files for different platforms, with spacing modified to suit each environment.

4. Alphabetical class reference

To be written.

5. Bugs

Version 1.0

- No XView or Motif versions yet.
- In Watcom-compiled 32-bit Windows version, bitmap buttons and messages don't work properly.
- Some panel item properties missing, e.g. wxPASSWORD, wxREADONLY.
- When dragging a selected item, other selected items should follow (to be consistent with convention), but don't.
- wxSlider sizing a bit broken: a wxWindows problem.

6. Technical notes

6.1. Overview

The dialog editor is written as a library, to be invoked by other programs. As you can see, dialoged.cc is a very small program which invokes the main window via a wxResourceManager object. The wxResourceManager object controls the user interface and other aspects of the dialog editor.

There is wxResourceTable object in wxResourceManager: this contains a list of all the wxItemResources currently being edited. wxResourceTable and wxItemResource are classes already in wxWindows, defined in wx_res.h. In order to edit a new dialog box, the dialog is created, and the existing event handler is temporarily replaced with a new one which defines editing functionality. This allows existing dialogs - even instances of subclasses of wxDialogBox - to be edited, the application-specific functionality being temporarily taken over by the dialog editor.

In order to edit the properties of a dialog box or item, a property list editor is invoked. This uses the property classes from utils/wxprop. In order to map between properties and the actual window API, such as SetSize and GetSize, a 'proxy' class called wxPropertyInfo has been defined, with a subclass for each class of wxWindows window to be edited. This class defines the main members SetProperty, GetProperty, GetPropertyNames, which transform the normal API into 'property' terms.

Properties are mostly extracted directly from the window being edited. This is in contrast with wxBuilder, where everything is stored in a set of parallel data structures, and windows 'properties' only only set. However, there are exceptions to this rule in the dialog editor. There *is* in fact a set of parallel objects, the wxItemResource objects which can be seen listed in the main Dialog Editor window as a dialog is built up. These usually parallel the properties in the windows, but occasionally this is not possible. For example, all dialog boxes being edited must be modeless: or the user would not be able to access other windows. However, the user must be able to specify that when used in an application, that dialog box will be modal. In this case, the value in the wxItemResource will not match that in the actual dialog box.

There is a major problem with taking values directly from the windows: this information sometimes does not match what went in. In Motif and XView, size values returned are not the same as those given. This causes speedy 'degeneration' of window properties. Under Windows, properties are almost always consistent. The other platforms will need to be catered for by relying more on the wxItemResource objects, and not taking size information directly from windows.

6.1.1. Dynamic setting versus recreation

The property editor scheme relies on being able to set window properties dynamically: the user changes a value, and the window changes immediately to reflect the new value. Unfortunately, not all properties can be changed dynamically in wxWindows; for example, in Motif, the label position must be given at panel item creation time, because the way the widgets are laid out depend on the label position. The label position cannot then be changed without deleting and recreating the item.

Hence the dialog editor takes two approaches: where values are dynamically settable, this is done. Where they are not, the item is deleted and recreated, after all existing values have been transferred into the parallel wxItemResource object. Therefore in wx_rprop.cc, some of the SetProperty implementations have one or more call to RecreateWindowFromResource.

6.2. Resource associations

wxItemResource objects (containing information about panel items and dialogs) are not visual objects. However, they need to be associated with the visual objects when the latter are created for editing purposes. Therefore there is a hash table called resourceAssociations in wxResourceManager. When a window is created, the resource pointer and window pointer are associated via the hash table. When the window is deleted, the association is removed. Children of a dialog are associated with child wxItemResource objects by calling wxFindWindowByName with the wxItemResource name.

6.3. What needs to be done for XView and Motif

The following areas need attention before Dialog Editor will run properly on these platforms.

1. For XView, the property editor needs to be made a modeless, not modal dialog, which has implications for flow of control in wxPropertyInfo::Edit.
2. Properties which do not return the same value they are set to, such as width and height, need to be stored directly in wxItemResource and *not* transferred from window to wxItemResource in wxWindowPropertyInfo::InstantiateResource.
3. Properties which cannot be dynamically set in XView or Motif need to have the item recreated (e.g. labelOrientation).

6.4. Files

The Dialog Editor source files are as follows:

- wx_rprop.h, wx_rprop.cc: handle property setting and getting through the 'proxy' wxPropertyInfo classes and using the property list editor from utils/wxprop.
- wx_resed.h, wx_resed.cc: the main implementation, in particular the wxResourceManager class.
- wx_reswr.cc: resource writing code.
- wx_repal.cc: the dialog editor palette implementation.
- dialoged.h, dialoged.cc: small 'stub' for invoking the user interface via a wxResourceManager object.

Index