User Manual for Dialog Editor Version 2.1

Julian Smart, Anthemion Software

April 2000

Contents

Chapter 1 Introduction..1
Current status 1
Future developments 1

Chapter 2 Commands 2
Dialog editor menu bar 2
Command toolbar 2
Tool palette 3
Resource tree 3

Chapter 3 Procedures..4
Running Dialog Editor.. 4
Creating a dialog.. 4
Using property editors.. 4
Saving and loading files.. 5
Working with identifiers... 5
Multi-platform development 5
Converting old files 6

Chapter 4 Change log 7

Chapter 5 Bugs...9

Chapter 6 Technical notes..10
Overview.. 10
Resource associations.. 11
What needs to be done for XView and Motif 11
Files 11

Chapter 7 Index 12

Copyright notice

Copyright (c) 2000 Julian Smart, Anthemion Software

Please see the wxWindows licence for conditions of use.

Chapter 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 2.0 should be compiled and used with wxWindows 2.0.

Current status

Dialog Editor currently runs under wxMSW and wxGTK. It has yet to be tested under wxMotif.

Future developments

- Motif compilation.
- 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.

Chapter 2 Commands

Dialog editor menu bar

File menu

New Dialog	Creates a new dialog resource.
New Project	Creates a new project (clears index and resets project name).
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.
Convert Old Resources...	Takes a directory of wxWindows 1.68 dialog resources, and converts them to wxWindows 2 resources, in a separate directory. See *Converting old files* (p. 6).
Exit	Exits the program.

Edit menu

Test Dialog	Creates the current dialog for test purposes.
Recreate	Recreates the currently selected control 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.

Help menu

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

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 controls horizontally.

	Aligns the top sides of the selected controls horizontally.
	Aligns the bottom sides of the selected controls horizontally.
	Aligns the centre of the selected controls vertically.
	Aligns the left sides of the selected controls vertically.
	Aligns the right sides of the selected controls vertically.
	Copies the size of the first selected control to the subsequently selected control(s).
	Copies the width of the first selected control to the subsequently selected control(s).
	Copies the height of the first selected control to the subsequently selected control(s).
	Evenly distributes the space between the selected controls, horizontally. Note that the controls should be selected in order from left to right.
	Evenly distributes the space between the selected controls, vertically. Note that the controls should be selected in order from top to bottom.
	Puts the selected control(s) to the front of the display list.
	Puts the selected control(s) to the back of the display list.
?	Invokes Dialog Editor help.

Tool palette

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

Resource tree

The resource tree shows a list of the dialogs, controls and bitmaps currently loaded in Dialog Editor. Double-clicking on an item shows the associated resource.

Chapter 3 Procedures

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.

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

To move a control, drag the item with the left mouse button, or edit the position values in the property editor. To resize a control, 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.

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 control. 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.

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.

Working with identifiers

Dialog Editor keeps track of identifiers in your resources, and reads and writes an include file of the form `name.h` where 'name' is the root name of your `.wxr` file. Dialog Editor knows about the predefined identifiers such as wxID_OK.

When you create a dialog or control, the identifier is initially generated. When you edit the identifier via a property editor, you can choose a new name, such as a predefined symbol and optionally change the integer assigned to the name (assuming it's not a predefined symbol).

When you save the project, the identifier include file is saved as well. Include this file in your project so that you can refer to controls and dialogs by identifier rather than obscure integers. Note that the `.wxr` file itself can only contain integer ids and not the symbols, due to way in which the resource file is loaded.

Multi-platform development

`.wxr` files generated on one environment (e.g. Windows) can be used in another (e.g. GTK). If you use default fonts and colouring (set **useSystemDefaults** to True in the dialog properties) then the dialog fonts and colours will take on the native values, rather than ones specified in the resource. Without this, colours in the dialog resource may not match system colours.

Also, set **useDialogUnits** to True whenever possible since this will cause the dialog to be created using a scale based on the current system font size, and will result in dialogs that are portable between screen resolutions as well as platforms.

Because the same control 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. The best thing is to try your dialog resource on several platforms and see whether tweaking is required for some platforms.

Converting old files

Dialog Editor can make an attempt at converting dialog resources created with Dialog Editor for wxWindows 1.68. The command is **Convert Old Resources...** on the **File** menu.

You need to specify two directories, an input and an output directory. Dialog Editor will do the following conversions:

1. wxMultiText becomes a wxTextCtrl with wxTE_MULTILINE style.
2. wxText becomes a wxTextCtrl.
3. wxMessage becomes either a wxStaticText or wxStaticBitmap.
4. wxButton becomes a wxBitmapButton if necessary.
5. wxGroupBox becomes wxStaticBox.
6. Controls that no longer have labels, such as wxTextCtrl and wxListBox, have a separate wxStaticText control created for them at approximately the correct position. The label's window name becomes ControlName_Label where ControlName is the name of the control that formerly had the label.
7. Identifiers are allocated.
8. Font sizes are reduced to counter the decreased font size now created by wxWindows for a given point size.
9. The dialog height is reduced slightly to compensate for the fact that the dialog caption is no longer included in the size.

Chapter 4 Change log

April 22nd, 2000 Version 2.1

- Various bug fixes.
- Added buttons for distributing space horizontally and vertically, and for copying width and height independently.
- Added 'Convert old resources' facility.

December 31st 1998, Version 2.0

- wxWindows 2.0 port.
- Major user interface changes.
- Allows identifiers to be edited and reads/writes an id header file.

March 15th 1997, Version 1.7

- Added fix to wx_rprop.cpp to avoid Fafa bitmap buttons growing every time the button edited.
- Added fix to wx_resed.cpp, case wxID_EXIT, to clean up properly on exit, avoiding double deletion of wxBitmap.

May 6th 1996, Version 1.6

- Added panel editing in addition to dialog box editing.
- Cured some bugs with changing window styles such as wxUSER_COLOURS and label position.
- Now preserves syntax of bitmap resources in wxr files.

March 1996, Version 1.5

- Changed behaviour of New tool, and changed File menu to include New project and New dialog items. Behaviour should be more standard now.

March 1st 1996, Version 1.4

- Items (but not dialogs) can now have duplicate names.
- Can pass a filename to the program from the command line.
- Cured bizarre error caused by a Windows combobox sending a fake left-mouse-up error when losing the focus (switching to another window). This fix will be in wxWindows 1.66.
- Rewritten code to use only the new type system, and to take account of of new window style partitioning (flags for different items may have the same value). Again, wxWindows 1.66 will have the new style values, to make room for more window styles.

January 28th 1996, Version 1.2

- Now starts off in non-user-colour mode under Windows
- Dragging item drags other selected items
- wxMessage saves size correctly, if used in conjunction with wxWin 1.66

January 19th 1996, Version 1.1

- Cured crash bug when quitting dialog window
- Added Clear menu item
- Added window type name to property window

December 19th 1995, Version 1.0

- First release.

Chapter 5 Bugs

Version 2.0

- No Motif version yet.
- Some control properties missing.
- When dragging a selected item, other selected items should follow (to be consistent with convention), but don't.
- No grid.
- No keyboard shortcuts.
- No tab ordering.
- In dialog unit mode, controls will sometimes move slightly when properties are edited, because translating between units isn't always reversible (rounding errors?).

Chapter 6 Technical notes

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.

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.

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.

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).

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.

Chapter 7 Index

—D—

Dynamic setting versus recreation, 10

—E—

Edit menu, 2

—F—

File menu, 2

—H—

Help menu, 2