Scrolling Text User Item in Dialog Boxes

How to use the ScrollDialog Library.

17th January, 1993 © by Graham Cox.
version 1.2 31st August, 1993

Displaying a lot of text in a dialog box in a scrollable panel is sometimes required, for example in “About” boxes, or for on-line help systems. Doing this with the dialog manager is something of a chore, however, as it requires three custom procedures: A user item procedure to draw it, a custom filter to handle the scrollbar hits, and a custom control action procedure to implement continuous scrolling. This library sets all of this up in a very high-level way so that for most purposes, a single call is all that is necessary to display a dialog box with a scrollable text panel. For more complex dialog boxes, a lower level interface is provided which takes much of the drudgery out of this task, and does it in a clean, consistent way. Version 1.1 provides for editing the text in the panel in a general way, bundling many of the chores associated with this task into the library code.

High-level interface.

The high-level interface is a single call which will display a given dialog, set up a particular user item as a scrollable panel, and install resource based text into it. It then handles hits in the dialog, scrolling the text as necessary, and returning the item number of the enabled item that closed the dialog box. In this respect, it acts a little like an alert call, in that the dialog is modal, and is closed by any enabled item. For modeless, or more complex modal dialogs, you will need to use the lower level interface described later.

```
pascal int ScrollTextDialog(int dialogID,int textUserItem,
                                                    int     textResourceID);
```

The function accepts three integer parameters- the resource ID of the ‘DLOG’ template to use as the dialog, the number of the user item to be used as a scrollable text panel, and optionally the resource ID of a ‘TEXT’ resource which will be displayed in the panel. The function returns the item number of the enabled item which closed the dialog. In order to use this function, you must create a dialog template containing a user item, and usually some other control, such as a button, to dismiss the dialog. All other types of items are supported in the normal way. The scrollbar for the text panel is created for you at run time, there is no need to create a ‘CNTL’ item in the dialog template. If the text resource doesn’t exist, the panel will be empty, and it will be necessary to install the text yourself. If a ‘styl’ resource exists with the same ID number as the ‘TEXT’ resource, the text will be displayed according to the style information in the resource. The easiest way to create this styled text resources is with version 2.1 or later of ResEdit, which creates the ‘styl’ resource for you automatically. This function has not changed in version 1.1- this call will always create a non-editable text panel. For editable text, you must use the lower level interface.

Lower-level interface.

The lower-level interface consists of three calls: one to create the dialog, one to handle it modally, and one to dispose of it properly. For modeless dialogs, you will need to handle much of the hit testing, etc. yourself, but this is starting to get beyond the purpose of this library.

```
pascal DialogPtr GetTextDialog(int dialogID,
                               int textUserItem,
                               int textResourceID,
                               Boolean canWrite);
```

This call does most of the work. In version 1.1, the <canWrite> parameter has been added. If set to FALSE, the old behaviour is implemented. If set to TRUE, then the panel is editable. It builds a resource from a 'DLOG' template with the ID number passed, and creates the internal data structure which stores data about the scrollable panel. It then creates a TextEdit record to hold the text, and then installs a 'TEXT' resource with the given ID into it. <textUserItem> is the ID number of the user item that is used as the panel. If the template doesn't exist, the function returns NIL. If the given item isn't a user item, the dialog is returned, but the text isn't installed and the scrollable panel will be unimplemented. If the text resource doesn't exist, the panel is installed, but no text is displayed. You can install text yourself with the TDSetText call described below. You should normally set the dialog to be initially invisible, then position it and show it after using this call, but you are not required to do so. Note that the scrollbar is created by this function, and displayed to the right of the user item- you do not need to define a 'CNTL' item as part of the dialog template. Bear in mind that the scrollbar will require 16 pixels on the right of the user item. All of the text panel is contained within the boundary of the user item. For unstyled text, nine point geneva is used. For styled text, the font, size and style information is derived from the 'styl' resource.

When <canWrite> is set to TRUE, the panel is editable. The majority of this task is handled entirely automatically by ModalTDDialog, which has been modified to handle editable panels. Key presses are passed on to the text record, as are mouse clicks, etc. When editable, the return and enter keys affect the text and do not return the default button setting. The blinking cursor is automatically displayed, and the cursor is changed to an i-beam when it is over the text panel. The scrollbar will be adjusted automatically by changes arising within the scope of the panel- i.e. typing. If you want to make style changes to the text, this is possible, but you will need to ensure the scroll bar is correctly adjusted yourself. A call has been provided to allow you to do this easily. Dragging within the text will scroll the panel automatically.

```
pascal void DisposeTDDialog(DialogPtr theDialog);
```

Call when you are completely finished with the dialog created by the above function. All of the internal data structures, the scrollbar and the TextEdit record are disposed of, and then the dialog itself is disposed.

```
pascal void ModalTDDialog(int *theItem);
```

Call instead of ModalDialog for the scrollable panel dialog, if the dialog is modal. This is merely a structured call to ModalDialog which calls the internal filter function which handles the scrolling. The enabled item hit is returned in <theItem>. Apart from the scrolling, this call handles events exactly as ModalDialog does, as defined in Inside Macintosh. This function has been modified to handle text editing in the panel if

<canWrite> is set to TRUE.

```
pascal void TDSetText(DialogPtr theDialog,
                      Ptr text,
                      long tLength,
                      StScrpHandle stInfo);
```

This function allows any text in memory to be installed as text in the scrollable panel. theDialog is the pointer to the dialog, as returned by GetTextDialog above. text is a pointer to the start of some text in memory, and tLength is the number of bytes of text to install. stInfo is optionally a handle to a style record if the text is styled. Note that if the text was loaded as a handle, it should be locked immediately before making this call, and unlocked immediately afterwards. If stInfo is NIL, the text will be installed unstyled. The panel will be immediately redrawn with the new text, and the scrollbar reset to line 1, and the length of the text calculated so that the scrollbar will allow all of it to be scrolled into view. If the text fits entirely within the panel, the scrollbar will be disabled.

```
pascal void TDSetResourceText(DialogPtr theDialog,int textResID);
```

This functions is a structured call to TDSetText to install a 'TEXT' resource as the panel's text. <textResID> is the ID number of the 'TEXT' resource to load. If a 'styl' resource exists with the same ID, then the text will be styled using this info. This function creates a copy of the text, and the resource is released after installation.

```
pascal TEHandle GetTDTextHdl(DialogPtr theDialog);
```

This utility function returns the handle of the internal TextEdit record being used for the panel text. There is not normally any need to access this handle, but for unusual requirements, you may wish to do so. If you are editing text with version 1.1, obtaining this handle is a useful thing to do- as you will need to make direct calls to TextEdit to make style changes, etc, within the panel.

```
pascal void RecalTDBar(DialogPtr theDialog);
```

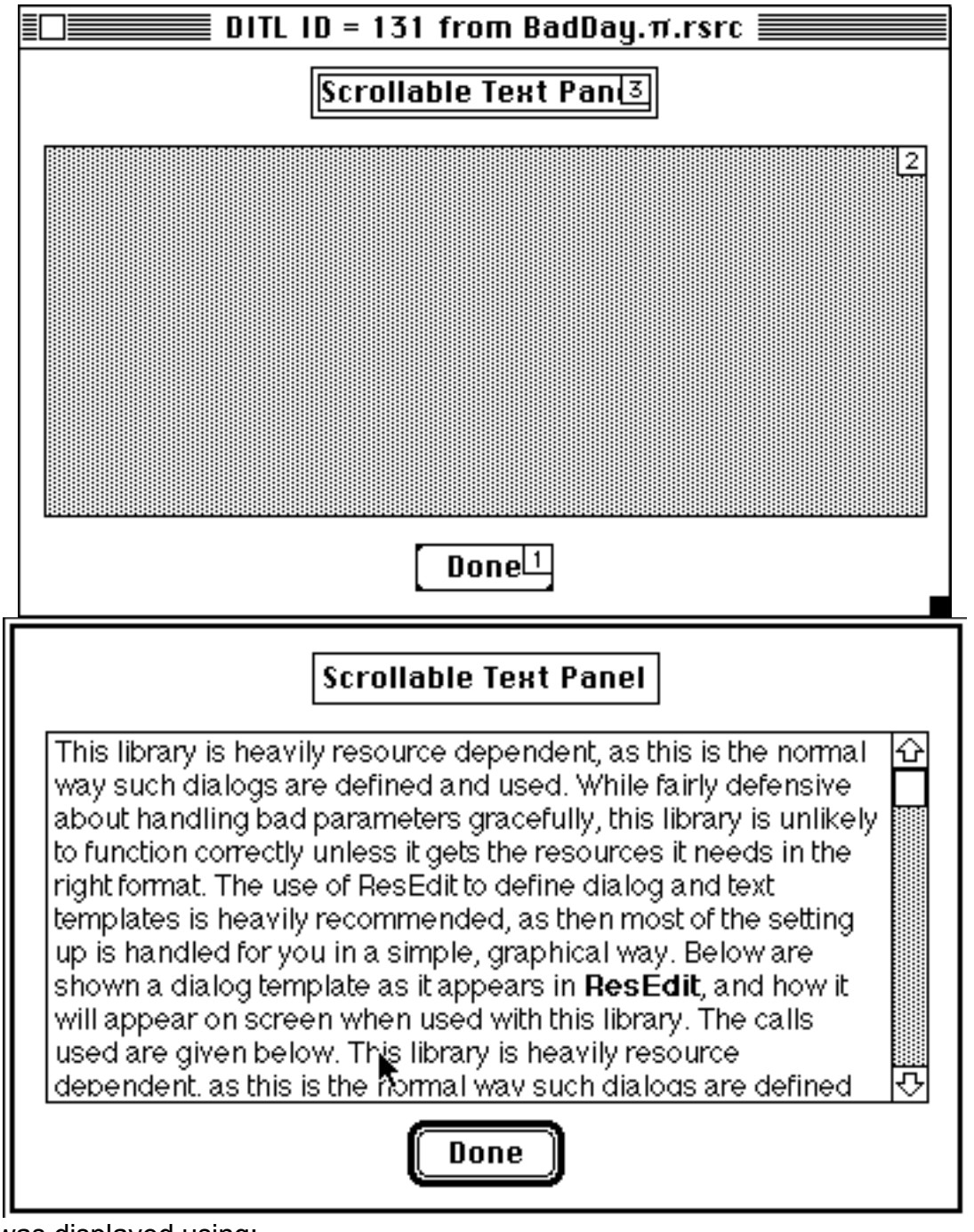
Available with version 1.1, this call recalibrates the scrollbar so that the full scroll range and position is correct after making an editing change, such as adding, deleting or changing the style of the text. You will only need to call this if you force changes from outside on the text panel, like a style change. Typing changes are automatically handled by ModalTDDialog.

In addition, version 1.1 fixes a small bug where the dialog's font characteristics got changed to geneva 9-point plain text for all subsequent calls to draw text items. The original font characteristics are now restored after setting up the text panel.

Resource set-up.

This library is heavily resource dependent, as this is the normal way such dialogs are defined and used. While fairly defensive about handling bad parameters gracefully, this library is unlikely to function correctly unless it gets the resources it needs in the right format. The use of ResEdit to define dialog and text templates is heavily recommended, as then most of the setting up is handled for you in a simple, graphical way. Below are shown a dialog template as it appears in ResEdit, and how it will

appear on screen when used with this library. The calls used are given below.



This was displayed using:

```
void ShowTextAlert(void)
{
int item;

item = ScrollTextDialog(131,2,129);
}
```

where the Dialog ID was 131, the user item was item number 2, and the text and style resources both had the ID 129. As you can see, the call is extremely simple.

Library Dependency.

This library is normally used in conjunction with the **xAlert.lib** library, and in fact *ScrollTextDialog* calls the *PosDialog* and *OutlineDItem* functions in that library, so these calls should be made available when using this library in your project, or you will get a link error. The only other dependency is that the Mac II version of TextEdit is required, which is available in System 6 or later.

Version 1.2 differs from version 1.1 in that all public functions now have pascal calling conventions. This allows the library to be used either in THINK C or THINK Pascal projects.