Simple Modeless Help System

How to use the SimpleHelp Library

10th February, 1994 © by Graham Cox
version 1.0

This document describes the API to the SimpleHelp library, which is a set of routines and data structures which allow you to add a simple help system to any application. This is a resource based system, using resources either included with the application or in a separate resource file. It is a multi-page text-only system, based upon the ScrollDialog library, which is also required to use this library. Either modeless or modal help dialogs can be used, though in most cases you will want a modeless one. Help is not editable in this implementation, the help system being set up either using ResEdit, or another program which I am planning which will be available in the future. Fig. 1. shows a very simple help screen that can be produced with this library.


Fig. 1.

This help library supports a simple type of hypertext (hot-links), which resolves a click in the text to a page ID, which is then installed. The presence of a hot link is detected by means of reserving a particular style as a hypertext style. This means that the user can recognise a hot link by the particular style, and expect to page switched to another page when such a styled item is clicked. The tables which are used to resolve particular words to a page ID are also defined as resources, and editing them is very straightforward.

High Level Interface.

This library builds on the ScrollDialog library, but inevitably is not as simple to use as that library partly because of the modelessness of the window, and partly because of the additional data structures required. n.b. it is important that an application has only one help window. This will normally be what you want, trying to open multiple help windows will result in confusion and possible instability. You should keep track of the handle referring to the help system in your main program, either as a global variable, or as a local in your main function. In any case, it is important that you have this handle available to determine if events should be passed to the help system for handling. Unlike some other libraries, this window is not readily identifiable by the program as anything special...

Creating the window:

```
SimHelpHdl      GetNewHelpDialog(       short DialogID,
                                        short listItem,
                                        short textItem,
                                        short topicID);
```

This function creates a data structure of type SimHelpHdl which is returned to your program. You must keep this value where you can access it from your event loop. The parameters to the function specify the resource ID's and item number ID's of various elements.

<DialogID> is the resource ID of the DLOG resource which specifies the dialog template for the help window.

<listItem> is the ID number of the USER ITEM which will be used for the topic list.

<textItem> is the ID number of the USER ITEM which will be used for the scrolling text panel.

<topicID> is the resource ID of the 'TPIC' resource which is described later, and contains the index (topic) strings plus all of the style, indentation and page ID information.

This function also creates the dialog window, the topic list, the scrolling text panel, and sets up a global variable with which it can find the list item within the drawing procedure in the library. This is necessary due to a limitation of the Macintosh and is the reason that only one help window can be defined.

Destroying a help window:

```
void      DisposeHelpDialog(SimHelpHdl sHelp);
```

Disposes of all of the data structures created above. Use once you have completely finished with a help window. The handle becomes invalid after this call. In normal use, you may prefer to just close the window and you can do this in the normal way with **HideWindow** and show it again with **ShowWindow**. To get at the WindowPtr (actually a DialogPtr) for the window, use:

```
DialogPtr          GetHelpDialogHdl(SimHelpHdl sHelp);
```

You can then pass this pointer to any dialog or window manager function as required. One note of caution- the ScrollDialog library stores data in the window's wRefCon field, so this field should not be used in your application.

Handling Events.

As the help window is modeless, you need to call **IsDialogEvent** for a newly received event, and then **DialogSelect**. If this function returns TRUE, it will also have returned the DialogPtr. You should then check this value against the one returned by **GetHelpDialogHdl**. If they are the same, you can then pass the event to:

```
pascal Boolean    ModelessHelp(SimHelpHdl sHelp,
```

```
                                    short theItem,
                                    EventRecord *theEvent);
```

This function will handle clicks in the topic list, scrollbars in both the topic list and text panel, and clicks on the ‘Prev’ and ‘Next’ buttons. It switches pages whenever the topic item changes, and also installs the current topic string as ParamText #1. If some other dialog item got clicked, such as any other button that you added to the template, the function returns FALSE. You can then do any additional dialog processing required.

Modal Help Window.

For VERY quick-and-dirty implementations, you can also provide a modal help window. However, this is not recommended (you users will curse you that they can’t move the window!). In this case, after you create you Help Window with **GetNewHelpDialog**, you should position it and make it visible, (if it isn’t already) and then call:

```
void      ModalHelpDialog(SimHelpHdl sHelp,short *theItem);
```

This will operate the dialog as a modal dialog, implementing all of the same operations as described above for the modeless case. It keeps control until the dialog is dismissed, which will happen if a button ID #1 is clicked, the return or enter keys are pressed, or an enabled item is clicked that was added to the template. In this case, the item number is returned in <theItem>.
After you have finished with the dialog, call **DisposeHelpDialog** as usual.

Special Requirements.

This library is designed to implement very SIMPLE help systems, that can be created in a few minutes and added to an application similarly quickly. If you require more sophisticated help, then perhaps it may be easier to write your own rather than trying to adapt this one. However, the remaining functions are available, and this documentation should allow you to use them if you wish.

```
OSErr             BuildTopicList(SimHelpHdl sHelp,Rect *listBox);
```

Called internally by **GetNewHelpDialog** to build the list data structure from the ‘TPIC’ resource. <sHelp> is the initialised handle, which should already have the dialog window created and initialised, and listBox is the rectangle of the list user item, normally obtained by calling **GetDItem**.

```
Boolean HandleHelpItems(SimHelpHdl sHelp,short *theItem);
```

Internal function to handle clicks in the list and the ‘Prev’ and ‘Next’ buttons. This function calls **SwitchPage** (described below) to change the page displayed according to the item

clicked. This function also handles the highlighting of the ‘Prev’ and ‘Next’ buttons to indicate whether the beginning or end of the topic list has been reached. It returns TRUE if the item was handled, FALSE if it was another item.

```
void      SwitchPage(SimHelpHdl sHelp,Cell clickCell);
```

Internal function to install a new page into the text panel. <clickCell> is the cell that has been clicked on in the topic list. This function makes some other checks: If the topic has a page ID of zero, the text panel is blanked completely. If the page ID specified by the topic cannot be located (the resource is missing), then this function installs the ‘missing page string’ into the text panel. **TDSetText** in the ScrollDialog library is called to install the text, which may be styled in accordance with the description of the functions in that library. Additionally, the text of the topic string is copied to ParamText #1 (^0) by this function. This allows you to add a title to the dialog window that changes according to the page. It is up to you to make provision for this string in your dialog template, and to make whatever calls are necessary after switching pages to update the string on screen.

```
void              InstallPage(SimHelpHdl sHelp,short pageID);
```

This function, which is called by **SwitchPage**, installs the given page into the text panel. <pageID> is the resource ID of the TEXT resource which contains the text of the help page.

```
pascal Boolean    ModelessText(DialogPtr theDialog,
                                                  short theItem,
                                                  EventRecord *theEvent);
```

This function allows the modal filter in the ScrollDialog library to be used in a modeless dialog to handle clicks in the text panel. It is used internally only, and it’s only possible use to an application programmer is to implement modeless windows containing text panels.

Hyperlink implementation.

Hyperlinks are implemented in a very simple way, and adding them to your help system involves very little work, as most of the processing is handled internally for you. Hyperlinks are identified within the help text by a particular style, or combination of styles. This style defaults to BOLD. Clicking on a word or group of words in this style causes the hyperlink table to be searched for that page. This table is a resource of type ‘Hypx’ which you add to your help resources. When an item in the hyperlink style is clicked, the text is scanned forward and backwards from the clicked point to locate the ends of the text in that style. This text is then highlighted, and checked against the entries in the hyperlink table. If a matching piece of text is found, the page ID associated with it is then passed to **InstallPage** to switch to that page. If the match wasn’t found, nothing happens. The following routines are available to support hyperlinks, but in most cases you will not need to use them, as all the required click processing, etc. is handled by **ModelessHelp**.

```
void              SetHyperlinkMatchStyle(SimHelpHdl sHelp,short theStyle);
```

This function should be called once after creating your help data structure to set the style for the hyperlinks. All possible styles and combinations of styles are allowable. If you do not call this function, the hyperlink style defaults to BOLD. If you do not want hyperlinks at all, then you should set the style to an unused style, unless you are sure that the text will not contain the default style.

```
short		GetHypertextStyleRef(TEHandle teRec,short theStyle);
```

This function returns the index number of the style as it is used within the TextEdit record for the current page. It is an internal maintenance function: you should never need to call it.

```
short		MatchHyperLink(Str255 *testStr,htIndexHdl pageIndex);
```

This function searches the hyperlink table for the page with the ID <pageIndex> for the occurrence of the string <testStr>. If found, the function returns the page ID of the matched page. If not found, the function returns -1. This functions looks for a resource of type 'Hypx' with the given ID. You should provide this resource for all pages that have hyperlinks in them. A ResEdit template for this resource type is provided. MatchHyperLink uses the toolbox function **EqualString** to determine if the string matched, and is not case sensitive. It also ignores diacritical marks.

```
short		ResolveHypertextLink(SimHelpHdl  sHelp,Point clickPt);
```

This function, which is called by **ModelessHelp** whenever **HandleHelpItems** returns FALSE, determines if the given click occurred in the help text. If it did, it checks to see if it occurred in a piece of text of the style defined for a hyperlink. If so, it locates the start and end of that text, highlights it, recovers the text itself, then calls **MatchHyperLink** to look up the page ID. It then returns the page ID to the caller, after unhighlighting the text. The caller is responsible for switching to the returned page. **ModelessHelp** does this by calling **InstallPage**.

Resource set-up.

This library depends, as usual, entirely on resources. Resources define both the appearance and layout of the help window, with 'DLOG' and 'DITL' resources, but also implement the entire help system, using 'TPIC', 'TEXT' and 'styl' resources. Additionally, if you have hyperlinks, you require a resource of type 'Hypx' for every page of text that contains a hyperlink. ResEdit templates for TPIC and Hypx resources are supplied with this library.

Dialog Template.

Though this library is intended to be flexible, some constraints are necessary. Two user items are required, one for the text panel and one for the topics list. In both cases, the scrollbar for the item is created for you at runtime on the right hand side of the item. The

topic list, and the text panel, including their scrollbars, fit within the user item's rectangle. These items can have any ID number, because you pass these ID numbers to **GetNewHelpDialog**. Three buttons should be provided- an 'OK' button ID #1 (if you have a modal dialog; for modeless ones you should create a dummy item #1). The 'Prev' and 'Next' buttons MUST HAVE ID's 2 and 3 respectively. Other than that, you can add any other items to the dialog that you require. If you add other user items, you should install their proc's after the call to **GetNewHelpDialog** and before you make the dialog box visible.

Text Pages.

Text pages are simply 'TEXT' resources, created in the normal way. ResEdit allows you to create styled text, creating the 'styl' resource for you. The ScrollDialog library knows how to handle both styled and unstyled text.

Topic Index.

This resource is the heart of the system. This uses a 'TPIC' resource which is defined by this library. ResEdit templates are provided for editing this resource type. The fields fields as displayed in ResEdit are shown in fig. 2.

 fig. 2.

<Indent> is the amount of indentation to apply to the entry. This is used to give a hierarchical appearance if desired. 0 is flushed left, 1 is one indent space, etc. In this version, an indent space is 10 pixels.

<Style> is the text style for the entry. 0 is plain text, 1 is bold, 2 is italic, 4 is underline, 8 is outline, 16 is shadowed. Values can be added together for combinations of styles.

<Font> is not currently used in this version. All entries use the default font which is 9pt Geneva.

<Page ID> is the resource ID of the TEXT resource which is the help entry for this topic. If this is set to zero, a blank page will be displayed

<topic String> is the topic entry text which will appear in the list.

Because this resource is implemented as a series of similar elements, it is simply a matter of using ResEdit to add or insert entries in the TPIC resource, or indeed delete them.

If a page cannot be found, the 'page missing string' is displayed. This is a resource of type 'STR ' which you should supply. This can be set to any suitable explanatory message. By default, this string should have an ID of 128, but this is stored in the

SimHelpHdl data structure and may be changed after creating the dialog to whatever ID you prefer.

Restrictions.

A TPIC resource should not contain more than 32K of data, or the list cannot manage it. Each page of text should also not exceed 32K, or TextEdit, which is used to display the text, will not manage it. It is very unlikely that you would have that much text on a page- if you find you have a large amount on some pages, consider breaking it down into sub-topics and spreading it over several entries, perhaps indented to show their relation under a common heading. Headings can have blank pages if required.

'Hypx' data structure.

The ResEdit template for this resource is shown below in fig. 3.

 fig. 3.

Like the TPIC type, it is a repeating resource, with as many entries as necessary, one for each hypertext link on the page. <Link Page ID> is the ID number of the page which is switched to when the link is clicked. <Match String> is the text of the hyperlink. When setting up your hyperlinks (which is best done as a last step in the creation of a help system), it is important to make sure that the style covers only the exact extent of a word or words that you are trying to match, or the link will not function because the match wasn't exact. For example, make sure that any spaces following a hyperlink word are set to plain style, or something else. The easiest way to ensure this is to double-click the word to select it, then apply the style.

SimHelpHdl data structure.

```
typedef struct
{
        WindowPtr       ownerWindow;    // window we are displayed in
        ListHandle      topicList;      // list of topics
        short           topicID;        // resource ID of topic list
        short           indentPixels;   // number of pixels per indent value
        short           listItemID;             // ID number of the list item
        short           textItemID;             // ID number of the text item
        Cell            lastSelection;  // remember which one we last displayed
        short           missingPageID;  // ID of 'missing page' string
        short           fontID;         // font to use
        short           fontSize;       // size of the font
        short           hyperTextIndex; // index of hypertext in styl run table
        short           hyperTextStyle; // style of hypertext link
        short           curPageResID;   // resource ID of the current page
        long            reserved;
}
SimHelpRecord, *SimHelpPtr, **SimHelpHdl;
```

Note that the TDRecord used by the text panel is not available through this data structure directly. Refer to the documentation for the ScrollDialog library if you need to get this record. (You shouldn't need it...)

Defaults.

```
#define         defaultFont                     geneva
#define         defaultFontSize         9
#define         defaultIndent           10
#define         defaultTextResID        128
#define         helpNextButton          2
#define         helpPrevButton          3
#define         pageMissingStrID        128
#define         topicListResType        'TPIC'
#define         hyperTextIndexType      'Hypx'
```

Function Summary:

```
SimHelpHdl      GetNewHelpDialog(short DialogID,
                                 short listItem,
                                 short textItem,
                                 short topicID);
OSErr           BuildTopicList(SimHelpHdl sHelp,Rect *listBox);
void            ModalHelpDialog(SimHelpHdl sHelp,short *theItem);
void            DisposeHelpDialog(SimHelpHdl sHelp);
DialogPtr       GetHelpDialogHdl(SimHelpHdl sHelp);
pascal void     ListDrawProc(WindowPtr theWindow,short theItem);

pascal void TopicListDef(short lMsg,
                         Boolean lSelect,
                         Rect *lRect,
                         Cell lCell,
                         short lDataOffset,
                         short lDataLength,
                         ListHandle theList);
```

```
pascal Boolean    ModelessHelp(SimHelpHdl sHelp,
                               short theItem,
                               EventRecord *theEvent);
void              SwitchPage(SimHelpHdl sHelp,Cell clickCell);
void              InstallPage(SimHelpHdl sHelp,short pageID);
Boolean           HandleHelpItems(SimHelpHdl sHelp,short *theItem);
pascal Boolean    ModelessText(DialogPtr theDialog,
                               short theItem,
                               EventRecord *theEvent);
pascal Boolean    TDFilter(DialogPtr theDialog,
                               EventRecord *theEvent,
                               int *itemHit);

short             GetHypertextStyleRef(TEHandle teRec,short theStyle);
short             ResolveHypertextLink(SimHelpHdl  sHelp,Point clickPt);
short             MatchHyperLink(Str255 *testStr,htIndexHdl pageIndex);
void              SetHyperlinkMatchStyle(SimHelpHdl      sHelp,
                                         short theStyle);
```

Note that only functions declared as ‘pascal’ can be called from a pascal program. This library is intended to be used with THINK C, and its use from a pascal program is not possible.

Dependencies.

This library depends on the following modules:

ScrollDialog.lib -implements the text panel.
xAlert.lib - alert and dialog utilities, including positioning functions, user item installer functions, other mundane but necessary stuff.
SneakyJump.c - this source file implements the **GetFunctionHandle** function which allows xDEF’s (like the LDEF in this library...) to be buried in the code and not as a separate resource.

How to use it in your application.

Include the SimpleHelp.lib library file in your project, and all of the libraries and files it depends on, listed above. #include the header file (SimpleHelpTypes.h) wherever you need to refer to a file in the library. That should be all that is needed.

Stuff.