Implementing Context Sensitivity in WinHelp and HTML Help

by Paul A. O'Rear of Helpful Solutions
www.helpfulsolutions.com pao@helpfulsolutions.com

The Controversial Part...
(...or the inspiration for this session)

- Lack of communication/ integration of development team with the electronic documentation team.
- A need for Help Authors to know enough about how the Context Sensitive Help system works, so they are not at the mercy of reluctant developers.
- A need for Developers to know enough about how the Context Sensitive Help system works, so that they can realize the demands of the tech writers are not unreasonable.

What Are We Going to Cover?

- Overview of "The Way Windows Works"
- What every help author should know about how Context Sensitive Help works
- Implementing Context Sensitive Help from the Help Author's perspective
- Implementing Context Sensitive Help from the programmer's perspective

The Way Windows Works

- events
- messages
- messages
- events
- events
- messages
- … you get the idea

Examples

- Event! - Click a button.
- Message! - A button was clicked. (WM_LBUTTONDOWN)
- Event! - Select a menu item.
- Message! - A menu item was selected. (WM_COMMAND)
- Event! - Minimize a window.
- Message! - A window was minimized. (WM_SIZE)

DON'T JUST SIT THERE — DO SOMETHING!

What Every Help Author Should Know About How Context Sensitive Help Works

- Five common Context Sensitive Help events.
- The five events and their corresponding messages.

Five Common Context Sensitive Help Events

- Help button click.
- F1 key press.
- “What’s this” (?) button click.
- Right mouse click Context Menu.
- Shift+F1

The Five Events and Their Corresponding Messages

- Help button click = WM_COMMAND
- F1 key press = WM_HELP
- “What’s this” (?) button click = WM_HELP
- Right mouse click Context Menu = WM_CONTEXTMENU
- Shift+F1 = WM_HELP

So What Messages Does the Developer Need to Handle?

- WM_COMMAND
- WM_HELP
- WM_CONTEXTMENU

Implementing Context Sensitive Help from the Help Author's Perspective

- WinHelp issues
- HTML Help issues
- Include file tips
- Food for thought...

WinHelp Issues

- topic ids (#) = Help Context ids in programmer’s include file (.h)
- [MAP] section entries vs. an include file.
- Help Compiler Workshop features and tips for error checking.

HTML Help Issues

- ***HASSLE!*** — Topics are individual .htm files — but — name your .htm files according to the programmer’s include (.h) file.
- ***BUG!*** — The HTML Help compiler only accepts include files that end with an .h extension.
- HTML Help only supports plain-text context sensitive help popups at this time.
- HTML Help also supports a unique Resource DLL approach to text popups which allows you to store the popup text as string resources — may aid performance.

Include File Tips

- Plan early on for a mnemonic approach to control and Help ID naming; e.g. IDH_OPTIONSDLG_FORMATBTN (cf. Nancy Hickman).
- DBHE.EXE — Helps automate creation of a list of items to document as well as an include file for you and the developer to use.
- Delphi and VB developers could save their forms as text and then extract the relevant information for an include file.

Food For Thought...

- The program doesn't have to wait for the WM_XXX messages — you could design your own conditions and events and send the WM_ messages yourselves...

Implementing Context Sensitive Help from the Programmer's Perspective

- The Help button
- The F1 key
- The "What's This" (?) button
- The right-click context menu
- The Shift+F1key

The Help Button

- Handle the WM_COMMAND message and issue a simple WinHelp() or HTMLHelp () API call.

WinHelp(hwndBtn, "ourhelp.hlp", HELP_CONTEXT, IDH_THIS_DLG_OVERVIEW);

HtmlHelp(hwndBtn, "ourhelp.chm", HH_HELP_CONTEXT, IDH_THIS_DLG_OVERVIEW);

The F1 Key

1) Create a DWORD array of the control ids and help ids.

```
static DWORD ids[] = {
      ID_SAVE, IDH_SAVE,
      ID_DELETE, IDH_DELETE,
      ID_COPY, IDH_COPY,
      ID_PASTE, IDH_PASTE,
      0, 0
      };
```

2) Handle the WM_HELP message and call the WinHelp() or HtmlHelp() API functions casting the lParam value passed with the WM_HELP message as a pointer to a HELPINFO structure:

```
WinHelp(((LPHELPINFO) lParam)->hItemHandle, "ourhelp.hlp",
  HELP_WM_HELP, (DWORD)(LPVOID) dwArray);

HtmlHelp(((LPHELPINFO) lParam)->hItemHandle, "ourhelp.chm",
  HH_TP_HELP_WM_HELP, (DWORD)(LPVOID) dwArray);
```

The What's This (?) Button

- ISSUE: The What's This button is only supported by Windows in dialog boxes where there's no minimize or maximize buttons.
- Enable the DS_CONTEXTHELP dialog style for the dialog.
- Handle the WM_HELP message as described for the F1 key.

The Right-Click What's This Context Menu

- Handle the WM_CONTEXTMENU message.
- Create a DWORD array as described earlier, cast the wParam value of the WM_CONTEXTMENU to an HWND and issue a call to either the WinHelp() or HtmlHelp() Windows API functions:

```
WinHelp((HWND)wParam, "ourhelp.hlp", HELP_CONTEXTMENU, (DWORD)(LPVOID)
   dwIDs);

HtmlHelp((HWND)wParam, "ourhelp.chm", HH_TP_HELP_CONTEXTMENU,
   (DWORD)(LPVOID) dwIDs);
```

The Shift + F1 What's This Functionality

- Handle the WM_HELP message as described for the F1 key.
- Depending on the development environment you may need to create a handler for the WM_KEYDOWN message that would handle Shift+F1 and then pass processing of messages to a message hook function until either the Escape key is pressed or a WM_HELP message is received. Process the WM_HELP message and then release the message hook.

Q&A

www.helpfulsolutions.com pao@helpfulsolutions.com