Context Sensitive Help Concepts

by Don Lammers

Last updated 2000-06-01

Presented with author's permission by

Shadow Mountain Tech

Contents

Scope

This paper is intended to show how I think online help for applications should look to the user. It is not intended to show how to implement these concepts, as implementation would be different for each programming language.

The following is strictly my own opinion. Since each programming language imposes its own restrictions on what can be done it may not always be possible to implement all of the suggested context sensitive help triggers. In addition, some may not be appropriate or necessary for your program or audience.

Basic Concepts

Working toward just-in-time, direct-to-brain information delivery.

- The closer you get the information to the user, the better.
- The less you bother the user, except when they need it, the better.
- Popups should be part of the navigation. They can lead the user into related topics. Microsoft does not do this, and does not even admit that it can be done, but it can and I do.
- The user should never need the TOC or Index. However, this does not excuse you from writing a good TOC or Index. Research done by Jared Spool and User Interface Engineering indicates that once the user resorts to searching on the Web, you know they are already lost.
- You should be as consistent as possible *from a user perspective* in how the context sensitive help is implemented. For instance, Microsoft implements context sensitive help differently in windows and dialog boxes. In windows you may get none, or there may be a **?** or  on the toolbar. In dialog boxes you have a **?** on the title bar. All the novice user knows is that things that look like boxes have help that doesn't work the same. Unfortunately this behavior is mandated by Windows, which will not allow the ? on the title bar if the window or dialog box is resizeable.

The UI as Help

Better yet, design an interface that doesn't need any user assistance. Although possible in theory, I have not seen one yet.

Traditional "Context Sensitive" Help

In Windows, the context sensitive help has come to mean the various ways in which the user can trigger online help from the program. The basic methods currently include using **F1** and **Shift-F1** to trigger help, a What's This? item in right click popup menus, a **?** button in the title bar of dialog boxes, and a  button in the toolbar. One typical way of implementing context sensitive help would be:

- Pressing **F1** opens dialog or window level help.
- Dialog boxes contain a **Help** button that opens dialog level help.
- Clicking the **?** button or  button puts the program into "What's This? Help" mode. Clicking a control opens a popup with a description of the control.
- Right clicking a control opens a popup menu with a What's This? item. Choosing this menu item opens a popup with a description of the control.

Various programming tools implement "standard" help in different ways. For instance, Visual Basic lets you choose between **F1** opening a window or dialog level help topic, or **F1** opening a control level topic. When given such choices, the most important thing is to be consistent throughout your program so that the user has some clue as to how help is triggered.

Embedded Help

Embedded help can often be the best context sensitive help (i.e., replacement for popups) that you can deliver. Unfortunately embedded help has some limitations (usually space) and a limited set of tools to help you through the process. WexTech's Help Extender is the only easy solution I know for this. If the user's system is guaranteed to include Internet Explorer, you can embed the WebBrowser control in your program and get a lot of flexibility without the need for add-ins.

One of the biggest advantages of embedded help is that it is always visible, so you can change the topics to match whatever is under the mouse pointer, or the user's current action. It also doesn't cover up part of the user interface.

How you implement embedded help depends on the application. The three main ways I've seen so far are listed below. You can combine these methods if it seems appropriate for your program and audience.

Instructional text appears next to each control, or each control includes a detailed description.

- This approach requires very careful layout so that expanding or shrinking text in localized versions doesn't encroach on the user interface elements.
- Text strings can be in the program resource file or in a separate text file that the program reads on startup. I prefer the latter approach as it is easier to separate the help authoring from the program.
- Each small text section is part of the user interface (probably a text box) so unless special arrangements are made the font size, layout, and such of each item are controlled by the program, not the author. Again, this information can be externalized in a style sheet, but this is additional work for the programmer.

Help is implemented as a window embedded in the main program window, and displays the main help topics.

- This approach takes up lots of real estate. You need to make sure that the user can "collapse" the embedded window, or perhaps even let them detach it.
- This approach is probably most useful for shorter topics--procedures and detailed descriptions of controls. You can always open a second window on top of the program for detail, or for theory topics.
- If the programmer uses the WebBrowser control (or an equivalent) to implement this, the topics can easily be authored in HTML, or even in a current help authoring tool with output to HTML Help or to straight HTML. Note that the WebBrowser control and HTML Help both require Internet explorer installed on the user's system.

Help is implemented as an embedded window which changes as you roll over the controls.

- This approach takes up some real estate. Again, you need to let the user choose to "collapse" the program window to gain back part of the real estate once they no longer need the help.
- Normally this approach is implemented to give the user more detail about the button or control the mouse pointer is currently over. You could easily add tabs for "Description" and "Procedure" to give the user the choice of whether they want to see a description of the control or how to use it.
- If the programmer uses the WebBrowser control (or an equivalent) to implement this, the topics can easily be authored in HTML, or even in a current help authoring tool with output to HTML Help (requires Internet explorer) or straight HTML.

How Basic Context Sensitive Help Should be Triggered

The following two scenarios show how I think context sensitive help should be triggered. This is strictly my own opinion. The first uses only traditional triggering methods, and the second uses an embedded window. By "triggering" I mean what user actions are necessary to trigger the help. It is not the intent to show how to implement this functionality, as the implementation (and in fact how much you can implement easily) is dependent on the programming tool used.

Using Traditional Triggering Methods Only

The Help Menu

The **Help** menu should contain at least the following items:

- An item to open the **Help Topics** dialog box.
- An item to open help for the current window.
- An item to put the program into What's This? Help mode.

Window and Dialog Level Help

- **F1** should open help for the window or dialog box.
- Each dialog box should include a **Help** button which triggers the same help topic as **F1**.

Control Level Help

- **Shift-F1** should put the program into What's This? Help mode. Once in What's This? Help mode, the mouse pointer changes to an arrow question mark and the user will get the help topic for the next control clicked.
- Dialog boxes should all contain the ? button in the title bar. Windows should contain the ? button in the toolbar (if there is a toolbar). These buttons put the program into What's This? Help mode. This is not ideal, but Windows will not let you put a ? button in the title bar of a resizeable window (you can set the window properties to show the button, but it will not appear).
- Right clicking a control should open a popup menu with a What's This? item, which opens help for the control. If What's This? would be the only item on the popup menu, a right click should open help for the control directly.
- Disable F1 opening help for the control. You already have two ways to get this help to the user. If you choose to have F1 help for controls, make sure it works consistently. Unfortunately, the standard implementation in Visual Basic opens help for the control under mouse pointer, but with menus, you have to have the menu highlighted. Make all the controls work the same. Unfortunately, you may be stuck with some behavior if you use standard controls, since their functionality is controlled by the company that programmed them.
- In windows, particularly in programs involving random text (like programming environments) you can also have F1 open help for the word currently containing the text cursor. For instance, in Microsoft Visual Studio if you click in a function name and then click F1, you get a description of that function.

Using Traditional Methods with an Embedded Window

The Help Menu

The **Help** menu should contain at least the following items:

- An item to open the **Help Topics** dialog box.
- An item to open help for the current window.
- An item to put the program into What's This? Help mode.

Window and Dialog Level Help

- **F1** should open help for the window or dialog box.
- Each dialog box should include a **Help** button which triggers the same help topic as F1.

Control Level Help

- Moving the mouse over a control should put the help for that control in the embedded window.
- There should be some mechanism to let the user open or close the embedded window easily.
- If you implement tabs in the embedded window you can let the user choose which type of information about a control appears. For instance, one tab might display a description while another would display a procedural topic about how to do whatever it is you do with this button.
- In windows, particularly in programs involving random text (like programming environments) you can also have F1 open help for the word currently containing the text cursor. For instance, in Microsoft Visual Studio if you click in a function name and then click F1, you get a description of that function.
- Unfortunately you are stuck with the way common dialog boxes work, or must override their functionality, which may not be a trivial project for the programmers.

Giving Even More Help

Context sensitive help can do as much for the user as your imagination and programming/authoring time allow.

- The right click popup menu lets you put both program functionality and help at the users fingertips. For help, you may decide to offer the user the standard What's This? Help as well as other items (for instance, a short video relevant to the control or a wizard that walks the user through the procedure that this control initiates).
- Many error messages need additional explanation. Provide a **Help** button, or have the error call help directly.
- Many warnings can use both an explanation and the option to let the program correct the problem and proceed. You can implement such choices in the program or in help by using training card help. Implementing them in help gives the author more control over content.
- You can trigger popups, messages, tooltips, and even the main help window depending on user actions. For instance, in Microsoft Visual Studio as you type a function you get a tooltip showing the parameters for the function. This "reminder" often keeps me from having to stop and go look up the information.
- You can implement warnings that appear only when the program spots a particular series of actions, or perhaps the order of the actions. For instance, in several HP scanning products, the program will post a warning that appears if you try to adjust color before exposure.
- You can trigger wizards, coaches, and other user assistance outside the main help system depending on user actions.

When creating such extensions to context sensitivity, you should keep the following issues in mind.

- Give the user has some way to turn off these features. They may be helpful the first time but become annoying once the user knows the product.
- Minimize false triggering. For instance, in Word 97, the Paperclip man would appear with an offer to teach you how to format a letter every time you typed a colon at the end of a line (at least, that's what seemed to trigger it). This was annoying the first time it happened, let alone the 50th.
- Don't have too many messages. Users start ignoring things that happen frequently. One project I worked on had a help topic that opened when the program spotted particular combinations of settings. We decided that we would hold the number of such messages to 10 or less (I think we ended up with 7), and that we would take a very close look at the program itself if more became necessary. There are legitimate reasons for warnings when user make certain combinations of actions or settings, but make sure that the warnings are about combinations that you need to allow but most users would find objectionable, and not settings that simply should not be allowed in the first place.

Resources

All of the following plus links to other reference sites about online user assistance are available through the Shadow Mountain Tech Web site (www.smountain.com).

On the Help/Connecting page:

- Latest update of this document and other documents about connecting online help to your program
- *Programmer's Reference to WinHelp* (by Don Lammers and Paul O'Rear)
- Sample code, including API definition modules.
- David Liske's help subclassing tutorial and modules

On the Help/Bugs & FAQ page:

- *WinHelp 4.0 Unofficial Bug and Anomaly List* (currently maintained by Don Lammers)
- *WinHelp FAQ File* (by Charlie Munro)