DOWNLOAD PDF WRITE CLEAR USER-INTERFACE TEXT

Chapter 1 : user interface - Racket GUI, can not write in a text field - Stack Overflow

The first experience people have with your mobile app is the most critical. If they cannot get it working right away, they won't finish setting it up and won't come back.

In this example, the prompt in the Search box is formatted as italic text. Use sparingly to emphasize specific words to aid in comprehension. Not all fonts support italic, so it should never be crucial to understanding the text. Place at the end of supplemental instructions, supplemental explanations, or any other static text that forms a complete sentence. Question marks Place at the end of all questions. Unlike periods, question marks are used for all types of text. Exclamation points In business applications, avoid. Exclamation points are sometimes used in the context of download completion "Done! Commas In a list of three or more items, always put a comma after the next-to-last item in the list. Colons Use colons at the end of external control labels. This is particularly important for accessibility because some assistive technologies look for colons to identify control labels. Use a colon to introduce a list of items. Ellipses Ellipses mean incompleteness. Use ellipses in UI text as follows: Indicate that a command needs additional information. For more information, see Command Buttons. Indicate that text is truncated. Indicate that a task is in progress for example, "Searching Truncated text in a window or page with unused space indicates poor layout or a default window size that is too small. Strive for layouts and default window sizes that eliminate or reduce the amount of truncated text. For more information, see Layout. To show truncated text, let users resize the control to see more text or use a progressive disclosure control instead. Quotation marks and apostrophes To refer to text literally, use italic formatting rather than quotation marks. For quotation marks, prefer double-quotation marks " " ; avoid single-quotation marks. Capitalization Use title-style capitalization for titles, sentence-style capitalization for all other UI elements. Doing so is more appropriate for the Windows tone. For legacy applications, you may use title-style capitalization for command buttons, menus, and column headings if necessary to avoid mixing capitalization styles. This generic example shows correct capitalization and punctuation for property sheets. This generic example shows correct capitalization and punctuation for dialogs. For feature and technology names, be conservative in capitalizing. Typically, only major components should be capitalized using title-style capitalization. Analysis Services, cubes, dimensions Analysis Services is a major component of SQL Server, so title-style capitalization is appropriate; cubes and dimensions are common elements of database analysis software, so it is unnecessary to capitalize them. For feature and technology names, be consistent in capitalizing. If the name appears more than once on a UI screen, it should always appear the same way. Likewise, across all UI screens in the program, the name should be consistently presented. Address bar, Links bar. Instead, follow the capitalization used by standard keyboards, or lowercase if the key is not labeled on the keyboard. Studies have shown that this is hard to read, and users tend to regard it as "screaming. For more information, see the "Text" or "Labels" section in the specific UI component guidelines. The user selects these in the Region and Language control panel item. In these examples from Microsoft Outlook, both formats for the long date are correct. They reflect different choices users have made in the Region and Language control panel item. Use the long date format for scenarios that benefit from having additional information. While users choose what information they would like to include in the long and short formats, designers choose which format to display in their programs based on the scenario and the context. In this example, the long date format helps users organize tasks and deadlines. Globalization and localization Globalization means to create documents or products that are usable in any country, region, or culture. Consider globalization and localization when writing UI text. Your program may be translated into other languages and used in cultures very different from your own. For controls with variable contents such as list views and tree views , choose a width appropriate for the longest valid data. Include space enough in the UI surface for an additional 30 percent up to percent for shorter text for any text but not numbers that will be localized. Translation from one language to another often changes line length of text. Instead, use complete sentences so that there is no ambiguity for the translator. Such a design is not localizable because sentence structure varies with language. In the incorrect example, the text box is placed inside the check box label. Link

text should therefore form a complete sentence by itself. Glossary links can be inserted inline, as part of a sentence. For more information, see the Go Global Developer Center. Title bar text Choose the title bar text based on the type of window: Top-level, document-centric program windows: Use a "document name program name" format. Document names are displayed first to give a document-centric feel. Top-level program windows that are not document-centric: Display the program name only. Display the command, feature, or program from which the dialog box came. For more guidelines, see Dialog Boxes. Display the wizard name. Note that the word "wizard" should not be included in wizard names. For more guidelines, see Wizards. For top-level program windows, if the title bar caption and icon are displayed prominently near the top of the window, you can hide the title bar caption and icon to avoid redundancy. However, you still have to set a suitable title internally for use by Windows. These concepts are implied and leaving these words off makes the titles easier for users to scan. Main instructions Use the main instruction to explain concisely what users should do in a given window or page. Express the main instruction in the form of an imperative direction or specific question. Error messages, warning messages, and confirmations may use different sentence structures in their main instructions. Use specific verbs whenever possible. Enter your locale, region, and language Better: In this example, the context of the UI is already very clear; there is no need to add main instruction text. Be concise use only a single, complete sentence. Pare the main instruction down to the essential information. If you must explain anything more, consider using a supplemental instruction. If the instruction is a question, include a final question mark. For progress dialogs, use a gerund phrase briefly explaining the operation in progress, ending with an ellipsis. You can evaluate a main instruction by imagining what you would say to a friend when explaining what to do with the window or page. If responding with the main instruction would be unnatural, unhelpful, or awkward, rework the instruction. For more information, see the "Main instruction" section in the specific UI component guidelines. Supplemental instructions When necessary, use a supplemental instruction to present additional information helpful to understanding or using the window or page, such as: Providing context to explain why the window is being displayed if it is program or system initiated. Qualifying information that helps users decide how to act on the main instruction. Prefer to communicate everything with the main instruction if you can do so concisely. Instead, omit the supplemental instruction if there is nothing more to add. Use complete sentences and sentence-style capitalization. Control labels Label every control or group of controls. Text boxes and drop-down lists can be labeled using prompts. Progressive disclosure controls are generally unlabeled. Subordinate controls use the label of their associated control. Spin controls are always subordinate controls. Omit control labels that restate the main instruction. In this case, the main instruction takes the access key. In this example, the text box label is just a restatement of the main instruction.

DOWNLOAD PDF WRITE CLEAR USER-INTERFACE TEXT

Chapter 2 : user interface - Select folder and write in a textbox the output powershell - Stack Overflow

Write Clear Text and Messages The wording of the interface and its screens is the basic form of communication with the user. Clear and meaningfully crafted words, messages, and text lead to greatly enhanced system usability and minimize user confusion that leads to errors and possibly even system rejection.

A column by Janet M. In this edition of Ask UXmatters, our panel of UX experts discusses the need for the work of technical writers to be an integral part of the UX design process. To get answers to your own questions about UX strategy, design, user research, or any other topic of interest to UX professionals in an upcoming edition of Ask UXmatters, please send your questions to: The following experts have contributed answers to this edition of Ask UXmatters: My company sells complex products that have weird names like universal variable life insurance and inland marine insurance. The industry is highly regulated, and we have to speak about products in a specific way, accompanied by legal disclosures. However, the mentality of the UX team is that writers are an afterthought. Anything that involves people interacting with something is inherently part of the user experience. Your situation, sadly, is not uncommon. Ironically, these professionals are usually the first to point out usability issues. They need to understand how something works so they can describe it to somebody else. My response has been to bring TC in to work with the design team as early as possible. In the old days of waterfall, we would actually try to write the Help or manual first, as our initial attempt at designing the experience. How little could we write? As we worked to trim the manual, we were baking in good design. Can we design workflows through the experience that emphasize the most likely needs or desired outcomes? Quite often, today, training and manuals focus on helping users to improve their domain knowledge rather than navigate a specific application. Or you can have a lavish design and crappy content, and no one will be happyâ€”including the people looking at the bottom line for your company. So, my advice to you is: Stay until they kick you out. You are a designer, too. There are multiple UX-related professions and skillsets that deal specifically with contentâ€”including content strategy, technical communications, information design, and content writing. By definition, you sometimes have to display exact words, no matter how confusing they are. If you have a situation like this, create a simple, plain-language explanation of what the legalese is saying. Emphasize this more understandable text for users, while displaying, but doing your best to deemphasizeâ€”to whatever extent legal constraints allowâ€”the stuff that nobody can understand. A user interface can be more successful when user expectations are understood and the interface maps to them. A problem can be as simple as: Is it because the design made the path difficult to see or find? But the two together, working in concert, can ensure that, when users expend effort considering a choice, they make the intended choice. So, creating user experiences really is a team effort. Your problem with communicating how to use complex products in specific or legally mandated ways is sadly typical.

Chapter 3 : c# - Remove all previous text before writing - Stack Overflow

When writing user interface (UI) text for Windows Mobile devices, consider the following questions: Is the user interface text clear, concise, and contextually relevant for the audience? Does the text consistently follow the style, terminology, and tone and voice specified for the product?

Who your users are and what devices they use should deeply inform your decisions here. Set expectations Many interactions with a site or app have consequences: So be sure to let users know what will happen after they click that button before they do it. There are two ways to help lessen the impact of human error: Prevent mistakes before they happen Provide ways to fix them after they happen You see a lot of mistake-prevention techniques in ecommerce and form design. Buttons remain inactive until you fill out all fields. Pop-ups ask you if you really want to abandon your shopping cart yes, I do, Amazonâ€”no matter how much it may scar the poor thing. Anticipating mistakes is often less frustrating than trying to fix them after the fact. That said, sometimes you just have to let accidents happen. Give feedbackâ€”fast In the real world, the environment gives us feedback. We speak, and others respond usually. We scratch a cat, and it purrs or hisses depending on its moodiness and how much we suck at cat scratching. All too often, digital interfaces fail to give much back, leaving us wondering whether we should reload the page, restart the laptop, or just fling it out the nearest available window. So give me that loading animation. Make that button pop and snap back when I tap itâ€”but not too much. And give me a virtual high-five when I do something you and I agree is awesome. Just make sure it all happens fast. Over 10 seconds, a disruption. If you do use progress bars on your site, consider trying some visual tricks to make the load seem faster. The time to acquire a target is a function of the distance to and size of the target. This obviously has all kinds of implications for interaction and UI design, but three of the most important are: This is especially important with menus and other link lists, as insufficient space will leave people clicking the wrong links again and again. Make the buttons for the most common actions larger and more prominent. Place navigation and other common interactive elements, like search bars on the edges or corners of the screen. This last might seem counterintuitive, but it works because it lessens the need for accuracy: Obviously, you can reinvent the wheel all you wantâ€”but only if it actually improves the design. Pocket wanted to focus people on the reading experience, and not duplicate an existing hardware control, but the inconsistent placement caused new users to accidentally close and archive the article they were reading, rather than simply returning to their reading list as expected. Some Limits on our Capacity for Processing Information. So, whenever possible, limit the number of things a person needs to remember to use your interface efficiently and effectively. You can facilitate this by chunking information, i. That can mean masking the complexity of an application behind a simplified interface whenever possible. A popular example of a product failing to follow this law is Microsoft Word. Most people only do a few things in Wordâ€”e. This led to a concept called progressive disclosure, where advanced features are tucked away on secondary interfaces. This also happens to be a best practice for mobile design, where robust navigation is always a challenge. This is especially true for users of screen readers. Make decision-making simple Too much of the web screams at us: Video interstitials stop us in our tracks, forcing us to watch precious seconds tick oh-so-slowly by. This impacts almost everything we build:

Chapter 4 : Using Forms and Processing User Input

This page contains guidelines for Drupal module and theme developers to use when writing user interfaces text (e.g., buttons, labels, in-page help, descriptions below fields, and (error) messages.

You can create high-level abstract data types called classes to mimic real-life things. These classes are self-contained and are reusable. Writing your own graphics classes and re-inventing the wheels is mission impossible! These graphics classes, developed by expert programmers, are highly complex and involve many advanced design patterns. However, re-using them are not so difficult, if you follow the API documentation, samples and templates provided. I shall assume that you have a good grasp of OOP, including composition, inheritance, polymorphism, abstract class and interface; otherwise, read the earlier articles. I will describe another important OO concept called nested class or inner class in this article. There are current three sets of Java APIs for graphics programming: Most of the AWT components have become obsolete and should be replaced by newer Swing components. The best online reference for Graphics programming is the "Swing Tutorial" http: For advanced 2D graphics programming, read "Java 2D Tutorial" http: For 3D graphics, read my 3D articles. It consists of 12 packages of classes Swing is even bigger, with 18 packages of classes as of JDK 8. Fortunately, only 2 packages - java. Custom graphics classes, such as Graphics, Color and Font. AWT provides a platform-independent and device-independent interface to develop graphic programs that runs on all platforms, including Windows, Mac OS X, and Unixes. Containers, such as Frame and Panel, are used to hold components in a specific layout such as FlowLayout or GridLayout. A container can also hold sub-containers. In the above figure, there are three containers: A Frame is the top-level container of an AWT program. A Panel is a rectangular area used to group related GUI components in a certain layout. In the above figure, the top-level Frame contains two Panels. There are five components: In a GUI program, a component must be kept in a container. You need to identify a container to hold the components. Every container has a method called add Component c. A container say c can invoke c. A Frame provides the "main window" for your GUI application. To write a GUI program, we typically start with a subclass extending from java. Frame to inherit the main window as follows: A Dialog has a title-bar containing an icon, a title and a close button and a content display area, as illustrated. An AWT Applet in package java. Panel and ScrollPane Secondary containers are placed inside a top-level container or another secondary container. AWT provides these secondary containers: As illustrated, a Container has a LayoutManager to layout the components in a certain pattern. Label provides a descriptive text string. Take note that System. You could use a Label to label another component such as text field to provide a text description. The first constructor constructs a Label object with the given text string in the given alignment. Note that three static constants Label. CENTER are defined in the class for you to specify the alignment rather than asking you to memorize arbitrary integer values. The second constructor constructs a Label object with the given text string in default of left-aligned. The third constructor constructs a Label object with an initially empty string. You could set the label text via the setText method later. Similarly, the getAlignment and setAlignment methods can be used to retrieve and modify the alignment of the text. Declare the component with an identifier name ; Construct the component by invoking an appropriate constructor via the new operator; Identify the container such as Frame or Panel designed to hold this component. The container can then add this component onto itself via aContainer. Every container has a add Component method. Take note that it is the container that actively and explicitly adds a component onto itself, NOT the other way. In the case, the Java compiler will assign an anonymous identifier for the allocated object. You will not be able to reference an anonymous instance in your program after it is created. This is usually alright for a Label instance as there is often no need to reference a Label after it is constructed. Button is a GUI component that triggers a certain programmed action upon clicking. The first constructor creates a Button object with the given label painted over the button. The second constructor creates a Button object with no label. Disabled Button cannot be clicked. The getLabel and setLabel methods can be used to read the current label and modify the label of a button, respectively. We will describe Swing later. Event Clicking a button fires a so-called ActionEvent and triggers a certain programmed action. I will explain event-handling

later. TextField is single-line text box for users to enter texts. There is a multiple-line text box called TextArea. You can convert a String to a primitive, such as int or double via static method Integer. To convert a primitive to a String, simply concatenate the primitive with an empty String. It has a top-level container Frame, which contains three components - a Label "Counter", a non-editable TextField to display the current count, and a "Count" Button. The TextField shall display count of 0 initially. We shall do that in the later example. In other words, this class AWTCounter is a Frame, and inherits all the attributes and behaviors of a Frame, such as the title bar and content pane. Lines 12 to 46 define a constructor, which is used to setup and initialize the GUI components. In Line 13, the setLayout inherited from the superclass Frame is used to set the layout of the container. FlowLayout is used which arranges the components in left-to-right and flows into next row in a top-to-bottom manner. A Label, TextField non-editable , and Button are constructed. We invoke the add method inherited from the superclass Frame to add these components into container. In Line , we invoke the setSize and the setTitle inherited from the superclass Frame to set the initial size and the title of the Frame. The setVisible true method Line 42 is then invoked to show the display. In brief, whenever the button is clicked, the actionPerformed will be called. In the actionPerformed Lines , the counter value increases by 1 and displayed on the TextField. The constructor is executed to initialize the GUI components and setup the event-handling mechanism. The GUI program then waits for the user input. For example, if we insert the following code before and after the setvisible: You could have an insight of the variables defined in the class. LEFT" default ; text is "Counter" assigned in constructor java. AWTAccumulator In this example, the top-level container is again the typical java. It contains 4 components: The components are arranged in FlowLayout. The program shall accumulate the number entered into the input TextField and display the sum in the output TextField. Frame Line 5 - the top-level window container. In the constructor Line 13 , we constructs 4 components - 2 java. Label and 2 java. The Frame adds the components, in FlowLayout. The listener class this or AWTAccumulator needs to implement ActionListener interface and provides implementation to method actionPerformed. In event-driven programming, a piece of event-handling codes is executed or called back by the graphics subsystem when an event was fired in response to an user input such as clicking a mouse button or hitting the ENTER key in a text field. Call Back methods In the above examples, the method actionPerformed is known as a call back method. In other words, you never invoke actionPerformed in your codes explicitly. The actionPerformed is called back by the graphics subsystem under certain circumstances in response to certain user actions. Three kinds of objects are involved in the event-handling: The source object such as Button and Textfield interacts with the user. Upon triggered, the source object creates an event object to capture the action e. This event object will be messaged to all the registered listener object s , and an appropriate event-handler method of the listener s is called-back to provide the response.

Chapter 5 : 10 essential UI (user-interface) design tips | Webflow Blog

Many applications are let down by the quality of the 'writing' in their user interfaces: typically, poor spelling, grammar, inconsistent tone, and worse yet, "humour" are the usual offenders.

What this section covers: See How to Use Password Fields. JComboBox Can be edited, and provides a menu of strings to choose from. See How to Use Combo Boxes. JSpinner Combines a formatted text field with a couple of small buttons that enables the user to choose the previous or next available value. See How to Use Spinners. The following example displays a basic text field and a text area. The text field is editable. The text area is not editable. Alternatively, to compile and run the example yourself, consult the example index. You can find the entire code for this program in TextDemo. The following code creates and sets up the text field: It does not limit the number of characters the user can enter. To do that, you can either use a formatted text field or a document listener, as described in Text Component Features. We encourage you to specify the number of columns for each text field. The next line of code registers a TextDemo object as an action listener for the text field. The text returned by this method does not include a newline character for the Enter key that fired the action event. You have seen how a basic text field can be used. Because the JTextField class inherits from the JTextComponent class, text fields are very flexible and can be customized almost any way you like. For example, you can add a document listener or a document filter to be notified when the text changes, and in the filter case you can modify the text field accordingly. Information on text components can be found in Text Component Features. Before customizing a JTextField, however, make sure that one of the other components based on text fields will not do the job for you. Often text fields are paired with labels that describe the text fields. You can find the entire code for this program in TextFieldDemo. As you type characters in the text field the program searches for the typed text in the text area. If the entry is found it gets highlighted. A status bar below the text area displays a message whether text is found or not. The Escape key is used to start a new search or to finish the current one. Here is a picture of the TextFieldDemo application. To highlight text, this example uses a highlighter and a painter. The code below creates and sets up the highlighter and the painter for the text area. The following code highlights the found text, sets the caret to the end of the found match, sets the default background for the text field, and displays a message in the status bar. The code below shows how the message method is implemented. Other methods you are likely to call are defined in the JTextComponent class. See The JComponent Class for tables of commonly used inherited methods. The API for using text fields falls into these categories:

Chapter 6 : Text-based user interface - Wikipedia

I've already did a cool little CLI To-Do app in python, and now I'm trying to build a basic GUI around it. The main buttons, field in place, and yesterday I've figured out how to redirect the return.

A genuine text mode display, controlled by a video adapter or the central processor itself. This is a normal condition for a locally running application on various types of personal computers and mobile devices. If not deterred by the operating system , a smart program may exploit the full power of a hardware text mode. A text mode emulator. This usually supports programs which expect a real text mode display, but may run considerably slower. Certain functions of an advanced text mode, such as an own font uploading, almost certainly become unavailable. A remote text terminal. The communication capabilities usually become reduced to a serial line or its emulation, possibly with few ioctl s as an out-of-band channel in such cases as Telnet and Secure Shell. This is the worst case, because software restrictions hinder the use of capabilities of a remote display device. Under Linux and other Unix-like systems, a program easily accommodates to any of the three cases because the same interface namely, standard streams controls the display and keyboard. Also, specialized programming libraries help to output the text in a way appropriate to the given display device and interface to it. See below for a comparison to Windows. Escape sequences may be supported for all three cases mentioned in the above section, allowing random cursor movements and color changes. However, not all terminals follow this standard, and many non-compatible but functionally equivalent sequences exist. However, programmers soon learned that writing data directly to the screen buffer was far faster and simpler to program, and less error-prone; see VGA-compatible text mode for details. On the other hand, programs running under Windows both native and DOS applications have much less control of the display and keyboard than Linux and DOS programs can have, because of aforementioned win32 console layer. Mouse cursor in Impulse Tracker. A more precise cursor per-pixel resolution was achieved by regenerating the glyphs of characters used where the cursor was visible, at each mouse movement in real-time. Later, the interface became deeply influenced by graphical user interfaces GUI , adding pull-down menus , overlapping windows , dialog boxes and GUI widgets operated by mnemonics or keyboard shortcuts. Some of these interfaces survived even during the Microsoft Windows 3. For example, the Microsoft C 6. Later versions added the Win32 console as a native interface for command-line interface and TUI programs. The console usually opens in window mode, but it can be switched to full, true text mode screen and vice versa by pressing the Alt and Enter keys together. Full-screen mode is not available in Windows Vista and later, but may be used with some workarounds. In Unix-like operating systems, TUIs are often constructed using the terminal control library curses , or ncurses , a mostly compatible library. The ability to talk to various text terminal types using the same interfaces led to more widespread use of "visual" Unix programs, which occupied the entire terminal screen instead of using a simple line interface. Some applications, such as w3m , and older versions of pine and vi use the less-able termcap library, performing many of the functions associated with curses within the application. The program minicom , for example, is modeled after the popular DOS program Telix. Some other TUI programs, such as the Twin desktop, were ported over. The Linux kernel supports virtual consoles , typically accessed through a Ctrl-Alt-F key combination. Up to 64 consoles may be accessed 12 via function keys , each displaying in full-screen text mode. The free software program GNU Screen provides for managing multiple sessions inside a single TUI, and so can be thought of as being like a window manager for text-mode and command-line interfaces. Tmux can also do this. The proprietary macOS text editor BBEdit includes a shell worksheet function that works as a full-screen shell window. The free Emacs text editor can run a shell inside of one of its buffers to provide similar functionality. There are several shell implementations in Emacs, but only ansi-term is suitable for running TUI apps. The other common shell modes, shell and eshell only emulate command lines and TUI apps will complain "Terminal is not fully functional" or display a garbled interface. In embedded systems[edit] Embedded system displaying menu on an LCD screen Modern embedded systems are capable of displaying TUI on a monitor like personal computers. This functionality is usually implemented using specialized integrated circuits, modules, or using FPGA. Users could move the

cursor over the entire screen area, entering and editing BASIC program lines, as well as direct mode commands. The Corvus Concept computer of used a function key -based text interface on a full-page pivoting display. Another kind of TUI was the primary interface of the Oberon operating system as released in Screenshot of the desktop of an Oberon System showing an image and several text viewers. In contrast to the so far mentioned uses of text user interfaces, the Oberon system did not use a console or terminal based mode but required a large bit-mapped display on which text was used as primary target for mouse clicks. Commands of the form Module. Any text displayed on the screen could be edited and every command displayed in a text, which complied to the required syntax, could be clicked and executed. Any text with a bunch of commands could be used as a so-called tool text serving as a user-configurable menu. Even the output of a previous command could be edited and used as a command. This approach is radically different from the dialogue oriented command prompt and console menus described so far. Since it did not use graphical elements, but text elements, it was termed a text user interface. For a short introduction see the 2nd paragraph on p.

Chapter 7 : User Interface Text | Microsoft Docs

Our User-Interface (UI) Text Team began with an editor whose goal was to organize a team that could keep all UI text consistent and manageable as our company moved to a new development technology.

Creating a Project The first step is to create an IDE project for the application that we are going to develop. We will name our project NumberAddition. In the Categories pane, select the Java node. In the Projects pane, choose Java Application. Type NumberAddition in the Project Name field and specify a path, for example, in your home directory, as the project location. Optional Select the Use Dedicated Folder for Storing Libraries checkbox and specify the location for the libraries folder. Deselect the Create Main Class checkbox if it is selected. Building the Front End To proceed with building our interface, we need to create a Java container within which we will place the other required GUI components. We will place the container in a new package, which will appear within the Source Packages node. Enter NumberAdditionUI as the class name. NumberAddition package replaces the default package. Once you are done dragging and positioning the aforementioned components, the JFrame should look something like the following screenshot. While the JPanel is highlighted, go to the Properties window and click the ellipsis Click OK to save the changes and exit the dialog. You should now see an empty titled JFrame that says Number Addition like in the screenshot. Renaming the Components In this step we are going to rename the display text of the components that were just added to the JFrame. Double-click jLabel1 and change the text property to First Number:. Double-click jLabel2 and change the text to Second Number:. Double-click jLabel3 and change the text to Result:. Delete the sample text from jTextField1. You can make the display text editable by right-clicking the text field and choosing Edit Text from the popup menu. You may have to resize the jTextField1 to its original size. Repeat this step for jTextField2 and jTextField3. Rename the display text of jButton1 to Clear. Or you can click the button, pause, and then click again. Rename the display text of jButton2 to Add. Rename the display text of jButton3 to Exit. Your Finished GUI should now look like the following screenshot: Adding Functionality In this exercise we are going to give functionality to the Add, Clear, and Exit buttons. The jTextField1 and jTextField2 boxes will be used for user input and jTextField3 for program output - what we are creating is a very simple calculator. Making the Exit Button Work In order to give function to the buttons, we have to assign an event handler to each to respond to events. In our case we want to know when the button is pressed, either by mouse click or via keyboard. So we will use ActionListener responding to ActionEvent. Right click the Exit button. Note that the menu contains many more events you can respond to! The IDE will open up the Source Code window and scroll to where you implement the action you want the button to do when the button is pressed either by mouse click or via keyboard. Your Source Code window should contain the following lines: Your finished Exit button code should look like this: Right click the Clear button jButton1. We are going to have the Clear button erase all text from the jTextFields. To do this, you will add some code like above. Your finished source code should look like this: It is going to accept user input from jTextField1 and jTextField2 and convert the input from a type String to a float. It will then perform addition of the two numbers. And finally, it will convert the sum to a type String and place it in jTextField3. Click the Design tab at the top of your work area to go back to the Form Design. Right-click the Add button jButton2. We are going to add some code to have our Add button work. The finished source code shall look like this: If you get a window informing you that Project NumberAddition does not have a main class set, then you should select my. To run the program outside of the IDE: The location of the NumberAddition project directory depends on the path you specified while creating the project in step 3 of the Exercise 1: Creating a Project section. After a few seconds, the application should start. If double-clicking the JAR file does not launch the application, see this article for information on setting JAR file associations in your operating system. You can also launch the application from the command line. To launch the application from the command line: On your system, open up a command prompt or terminal window. At the command line, type the following statement: NumberAdditionUI is set as the main class before running the application. You can check this by right-clicking the NumberAddition project node in the Projects pane, choosing Properties in the popup menu,

and selecting the Run category in the Project Properties dialog box. The Main Class field should display my. How Event Handling Works This tutorial has showed how to respond to a simple button event. There are many more events you can have your application respond to. Go back to the file NumberAdditionUI. Right-click any GUI component, and select Events from the pop-up menu. Alternatively, you can select Properties from the Window menu. In the Properties window, click the Events tab. In the Events tab, you can view and edit events handlers associated with the currently active GUI component. You can have your application respond to key presses, single, double and triple mouse clicks, mouse motion, window size and focus changes. You can generate event handlers for all of them from the Events menu. The most common event you will use is an Action event. How does event handling work? Every time you select an event from the Event menu, the IDE automatically creates a so-called event listener for you, and hooks it up to your component. Go through the following steps to see how event handling works. These methods are called event handlers. Now scroll to a method called initComponents. First, note the blue block around the initComponents method. This code was auto-generated by the IDE and you cannot edit it. Now, browse through the initComponents method. Among other things, it contains the code that initializes and places your GUI components on the form. This code is generated and updated automatically while you place and edit components in the Design view. In initComponents , scroll down to where it reads jButton3. The ActionListener interface has an actionPerformed method taking ActionEvent object which is implemented simply by calling your jButton3ActionPerformed event handler. The button is now listening to action events. Generally speaking, to be able to respond, each interactive GUI component needs to register to an event listener and needs to implement an event handler. As you can see, NetBeans IDE handles hooking up the event listener for you, so you can concentrate on implementing the actual business logic that should be triggered by the event.

Chapter 8 : Reviewing User Interfaces :: UXmatters

I'm currently writing a program (in Racket) in which I use multiple tabs. To do so I use the "tab-panel%". For each tab I then make a new vertical calendrierdelascience.com someone clicks on a tab, my callback procedure is called and I change the children of the "tab-panel%" so that now the vertical panel of the tab (the user clicked on) is set as child of the tab-panel.

Validating Form Input A well-designed Web form often includes a client script that validates user input prior to sending information to the server. Validation scripts can check for such things as whether the user entered a valid number or whether a text box was left empty. Imagine that your Web site includes a form that enables users to compute the rate of return on an investment. You will probably want to verify whether a user has actually entered numerical or text information in the appropriate form fields, prior to sending potentially invalid information to your server. Beyond prompting users more quickly about input errors, client-side validation yields faster response times, reduces server loads, and frees bandwidth for other applications. The following client-side script validates user input in this case, the script determines whether an account number entered by the user is actually a number prior to sending information to the server: However, some older browsers do not support this method. A particularly advantageous way of carrying out server-side validation is to create a form that posts information to itself. For more information, see Interacting with Client-Side Scripts. The input is returned to the same file, which then validates the information and alerts the user in case of an invalid input. Using this method of processing and validating user input can greatly enhance the usability and responsiveness of your Web-based forms. For example, by placing error information adjacent to the form field where invalid information was entered, you make it easier for the user to discover the source of the error. Typically, Web-based forms forward requests to a separate Web page containing error information. Users who do not immediately understand this information may become frustrated. For example, the following script determines whether a user entered a valid account number by posting information to itself Verify. If you are using JScript for server-side validation, be sure to place a pair of empty parentheses following the Request collection item either QueryString or Form when you are assigning the collection to a local variable. Without parentheses, the collection returns an object, rather than a string. The following script illustrates the correct way to assign variables with JScript: Write "Your name and password are the same. Write "Your name and password are different. This means that for both VBScript and JScript, in addition to placing a pair of empty parentheses following the Request collection item, you will need to specify the index of the desired value. For example, the following line of JScript returns only the first of multiple values for a form element: Form "Name" 1 ; A common use of intranet and Internet server applications is to accept user input by implementing a form in your Web page. ASP includes the following two collections in the Request object to help process form information: Accepting form input from clients gives malicious users a chance to send potentially unsafe characters to attack your Web application in any of the following ways: Strings that are too long for you application to handle can cause your application to fail or write over existing data. Strings that contain invalid characters can cause your application to fail or perform unexpected actions. If your Web application is accessing a database, a carefully engineered string of characters can add or delete records from your database. A general method of protection against these kinds of attacks is to Server. Another method is to write a short function that tests form input for invalid characters. This tutorial uses the Server. However, more information can be found by reading chapter 12 of Writing Secure Code , and using Checklist: The Web server then displays the user input. Later in this module, you use this knowledge about forms to build a guest book application that uses ASP scripting. To complete this lesson, you perform the following tasks: Display a selection of button elements in a form. Display text box elements in a form, accept the user input from the form, and display the user input on the Web page. Buttons Forms can contain many different kinds of elements to help your users enter data. In this example, there are five input form elements called buttons. After the user enters information in a form, the information needs to be sent to your Web application. When a user clicks the button labeled "Submit" in your Web page, the form data is sent from the client to the Web page that

is listed in the ACTION element of the form tag. In this example, the Web page listed in the ACTION element is the same as the calling page, which eliminates the need to call another page. Copy and paste the following code in your text editor, and save the file as Button. View the example with your browser by typing http:

Chapter 9 : Is Technical Writing Part of UX? :: UXmatters

Unknown terms or phrases will increase cognitive load for the user. Do your best to avoid 'geek speak'. A safe bet is to write for all levels of readers and pick common words that are clearly and easily understandable to both beginning and advanced users. Below is an example of using jargon in.

Our User-Interface UI Text Team began with an editor whose goal was to organize a team that could keep all UI text consistent and manageable as our company moved to a new development technology. This shared vision united the UI Text Team, which comprises two writers, two UX designersâ€”one of them being meâ€”and the editor who founded the team. We knew what was currently inconsistent in our user interfaces, including terminology, spelling, the use of abbreviations, labeling, caseâ€”sentence or title caseâ€”and the use of colons in labels. We included links to the guidelines from our documentation style guides and procedures, so writers and editors also became aware of them. We also created a laminated cheat sheet containing the guidelines that should be most commonly applied, but are often overlooked. We posted this cheat sheet online for our internal teams, as well as making it available as a downloadable PDF. When creating this cheat sheet, we integrated the guidelines into a wireframe of a user interface that, on one side, included annotations that spelled out our UI capitalization standards and, on the other, guidelines for typeâ€”including labels, hint text, and menu items. That cheat sheet is now outdated and no longer in use, but for many years, it served as a reference for writers, designers, and testers throughout our global offices. Newly hired writers, editors, and UX designers received a copy of it as part of their orientation. Perhaps because of its simplicity and usability, our cheat sheet became very popular. In general, the efforts and accomplishments of our UI Text Team were well received. Keeping the Team Running Our UI-text guidelines are now called standardsâ€”making the distinction that, while a guideline provides guidance, people must follow a standard. Since that first year, our UI Text Team has evolved. Although our team is now down to three membersâ€”me, the editor, and one of the writersâ€”we work in tandem with a corporate terminologist, who is responsible for terminology issues at our company and works on internationalization issues. My role has changed over the years. Our UI-text guidelines are now called standardsâ€”making the distinction that, while a guideline provides guidance, people must follow a standard. For example, one UI-text standard dictates that all labels have closing colons. As our products have progressed to keep pace with the ever-changing world of software design and technology, we have adapted to these changes by revisiting our existing text standardsâ€”sometimes establishing new ones rather than being rigid and our standards remaining static. Our process of reviewing standards is not formal. We try to make decisions quickly to address real-world scenarios. First, we propose a problem to solve or a question to answer, then we refer to our existing standards, research possible solutions, and make a decision. During our research and analysis, we may refer to existing standardsâ€”as well as the research on which we based those standards existing UI-design patternsâ€”and the work from which we derived them external standards and guidelinesâ€”from standards organizations or other companies usability test resultsâ€”from testing the products that incorporate our standards whitepapersâ€”from various organizations review commentsâ€”from our peers in Internationalization, Accessibility, and Legal, as well as writers, UX designers, and domain experts The keys to success in making UI-text standards workâ€”or really any standardsâ€”is that the standards deal with real cases, whether big or small, and that you make decisions quickly. The Changing Role of the Writer Although our standards are now in place, communicating the role of the writer in the design process can still be problematic for a variety of reasonsâ€¦. Although our standards are now in place, communicating the role of the writer in the design process can still be problematic for a variety of reasons, including the following: There are product teams who still implement their own UI text. Most commonly, some writers feel overwhelmed by having to support design efforts, as well as produce the documentation for many products at once. Some writers themselves do not think they should have a say in the design process. For the work I do, working with writers has been one of the most important aspects of my job. The writer for a product knows the domainâ€”and the user interfaceâ€”better than most people who work on the product. The software that we create at my company is very complex. I also rely on the writers to review long labels that

make translation to other languages more difficult. When people working on a product do not consult their writer and use a long label name, we ensure that the writer gets the opportunity to review it. Continuous Evolution We have achieved success by implementing UI-text standards, involving writers during the UI design and development process, and including writers in design reviews. Communication and education are key to making improvements. We have achieved success by implementing UI-text standards, involving writers during the UI design and development process, and including writers in design reviews. Standards and processes can lead to better quality and greater efficiency. Succeeding with standards means being smart about what you standardize, being consistent where consistency matters, being judicious about what you make into a standard or process; and making sure standards are efficient, usable, and evolutionary.