Computers & the Humanities 281

Additional Useful Commands

Here are a number of miscellaneous commands which you will find useful in the upcoming assignments and projects.

Push/Pop

The command `push card` stores the identification of the current card in a special location in Revolution's memory. The command `pop card` retrieves that stored information and goes directly to that card. These are used keep a pointer/bookmark to a particular card. Then if you go wandering around from card to card as you please, you can always get directly back to the original card by `pop card`.

Lock/Unlock

Ordinarily Revolution constantly updates the screen as things happen in a stack. When you hide an object or show an object or go to a different card, the change shows up immediately. The command `lock screen` freezes the screen appearance and blocks any visible changes until an `unlock screen` occurs. The only exception to this is the cursor. It continues to move with the mouse and any changes to the cursor icon (like set cursor to busy) do appear. The command `unlock screen` unfreezes the screen and allows Revolution to update the appearance with whatever changes may have happened. It is good programming practice to always unlock the screen after you have locked it in a handler.

You can optionally specify a visual effect with the unlock screen command: `unlock screen with barn door open` The effect is applied as the screen is updated. This can be used to convey a smoother change or transition.

This pair of commands allows you to do some processing "behind the users' backs" without them seeing what's happening. They are particularly useful when you have something that takes several steps to accomplish (such as updating a card) and you want them to see only the final result.

Answer

The answer command creates a dialog box, and the user is given buttons from which to choose his/her response. A dialog box is quite useful for getting the user's attention, particularly since the box if imposed in front of the card, and the user can do nothing until the box is acknowledged and dismissed. The simplest form of the command requires the keyword `answer` followed by a prompt of some sort:

```
answer "Alert: Continuing will disable all previous functions!"
```

When given in this fashion, the prompt literal is displayed in the top section of the dialog box. A button labeled "OK" is positioned at the bottom to the right. When the user clicks OK, "OK" is returned in the `it` variable.

If you desire to create custom buttons for the user to choose in the dialog box, these can easily be set with scripting. Follow the syntax for the prompt from the answer command given above, then add "with" and whatever you desire to appear as a button in quotations after the "with":

```
answer "You haven't answered all the questions." with "Duh!"
```

If you want to give multiple buttons as choices, add "with" and the first button label in quotations, separate choices with "or" and add the next button label in quotations:

```
answer "You haven't answered all the questions." with "Duh!" or "Oops!" or "So what?"
```

The last button label you put in quotations will appear as the default button in the dialog box (meaning that if the user presses return, this button will execute). Whichever button the user clicks on will be returned in the `it` variable, which can then be used to perform certain functions.

Your limit on how many buttons you can specify in the dialog box is determined by the operating system (I found the limit to usually be seven, and you don't usually want *that* many). However, proper design dictates that no more than three choices should be presented in a dialog box. The number of characters that can be used in the prompt and buttons of the dialog box differs according to operating system. The prompt will fit as many lines as will fit within the dialog box size of the operating system you are on. Buttons will expand to fit the label you have specified (but the limit does depend on the operating system; 80 characters is about the maximum size I've found).

As mentioned before, this is an effective way of getting the attention of the user. It can also be configured to give the user an opportunity to input some information (to a degree) that can affect the operation of the application after the choice has been made.

Ask

Another type of dialog box is created using the ask command. This creates a dialog box with a place for the user to type a response. The simplest form of the command requires the keyword `ask` followed by a prompt of some sort:

```
ask "Please enter your student number:"
```

When given in this manner, the prompt is displayed in the top section of the dialog box. A field is provided for the user's response, and buttons labeled "OK" and "Cancel" are positioned at the bottom, with "OK" designated as the default. If the user clicks "OK" (or presses Return), the contents of the field are returned in the `it` variable. Clicking "Cancel" returns "Cancel" instead of the contents of the field.

If you desire to show the user a model for their response to the dialog box, follow the syntax for the ask command described above and add "with" and then a default response in quotation marks:

```
ask "What is your name?" with "LastName FirstName"
```

Whatever literal was typed in quotations after the "with" will be shown hilited in the hilite color in the field in the dialog box provided for the user. The user can now choose to either keep that response (and click OK or press return) or type in a different one of his own. The two buttons work as before.

The number of characters that can be used in the prompt and response of the dialog box differs according to operating system. The prompt will fit as many lines as will fit within the dialog box size of the operating system you are on. The response can also have multiple lines (the number depends on operating system), but only one line will be shown at a time. In order for the other lines to be seen, the down and up arrow keys must be used.

This type of dialog box is extremely useful in extracting specific information from the user, which information can then be used in several different fashions or employed in several different functions. The possibilities are limitless.

Do

The Do command will take a line of text and execute it as a command. Usually the line of text comes from a field or variable, much like `do field 12`, where if field 12 contains the text "put 16 into thisNum". A do command can execute a line you type as a command in a handler. In fact, the do command will do any number of lines (or is supposed to; it's rather hit and miss, unfortunately). This is a useful way of overcoming the ten-line limit of imposed on scripts in the trial version of Revolution.

Grab

The Grab command allows the user to "grab" an object with the browse tool and move it to another location. When used in conjunction with some specific properties of objects, this is particularly useful with a drag and drop type of activity.

Course Schedule
Main Page