**Computers & the Humanities 281**

# Culture Readings Applications

Language textbooks will often contain Culture Reading sections: a selection of text about the target country or culture, usually accompanied by supplementary related material, be that additional text, illustrations, or pictures. We can emulate that function and even enhance it with the use of computers, where animations, sound, and video can be used in addition to the pictures and text. This is the essence of hyperlinking. This activity will focus on the mechanics of getting the text operate in the fashion we need.

## Spanish Text

Let's start with a selection of Spanish text in a scrolling text field. Our goal is to have certain words within that text clickable or "hot" so that the user will be taken to a glossary of sorts. The glossary can be either data fields on the card itself, or a separate stack (or any of a half-dozen other methods).

### Beginnings

First of all, we have to decide what we want to accomplish and think through the various possibilities and requirements. Working with any type of language is tricky due to the number of inflections of certain words. We want to make sure we acount for those. For the glossary, we would create a card separate from the readings with three fields:

- Key: the dictionary form of the word
- Definition: the English definition
- Inflections: the inflected forms of the key word (plurals of nouns, conjugated verb forms, etc.)

When designed as a separate stack, the Glossary stack would consist of only one card per key word. Otherwise, the data for each of the fields would be contained in three separate data fields.

Ideally we want to allow the student to click on any word in the text and view the corresponding information in the glossary. This would be facilitated by placing a handler in the field's script (don't forget to lock the text of the field):

```
on mouseUp
  put the clickText into whichWord
  select the clickChunk
  set cursor to watch
  push card
  lock screen
  go card "SpGlossary"
  find word whichWord in fld "keys"
  if the result is "Not Found" then -- word was not found
    find word whichWord in fld "inflect"
    if the result is "Not Found" then -- word not found again
      pop card
      answer error "The word '" & whichWord & "' was not found in the glossary."
      exit mouseUp
    end if
  end if
  put word 2 of the foundLine into thisLine
  put line thisLine of field "keys" into field "key"
  put line thisLine of field "inflect" into field "inflections"
  put line thisLine of field "glossary" into field "definition"
  unlock screen with zoom open very fast
end mouseUp
```

The use of the `clickText` function in the first line of the handler gives us the text of the word clicked and puts it into a variable. The user is then taken to the glossary where a search for the text is performed, restricting the search to key fields. A successful search will display the desired information. The text clicked is selected to give some visual feedback to the user to indicate that their click was registered.

Now comes the question of how to get back to the text after a successful search. The `push card` statement in the handler preserves card's ID in memory. Then we need a button with the glossary with this handler in its script:

```
on mouseUp
   pop card
end mouseUp
```

This will retrieve the ID of the card with the text and return the user to that card. When set up in this manner, we could have several readings that reference the glossary and the button will return them to the proper text of origin each time.

We have to account for the inflected forms. Hence the two `find` commands and nested `if...then` statements. The syntax of the find command doesn't allow more than one restriction, i.e., we can't script `find word whichWord in fld "Key" or "Inflections"`. We could also set the **Find command ignores** property to true for each field in the stack (usually practical only when the glossary is a separate stack).

It is good design to account for all probable possibilities, so we should always plan for the possibility the word will not be found, even if we think they should all be there. The if control statement in the field's handler will trap all unsuccessful searches. A number of Revolution commands return an error message if unsuccessful. Remember that if the search is unsuccessful, `not found` is placed in `the result`. If this function is empty, then that is an indication of a successful search. In this case we don't want to stay in the glossary (looking at the wrong word) so we `pop card` to return to the text and give an informative word-not-found message.

We also spare the user having to watch while we locate the word in the glossary. We can `lock screen` before going to the next card, and then `unlock screen` after we arrive there. We can also add a `zoom open` visual effect to give the appearance that we zoomed right there (as the zoom originates from the click location).

### Link Text

You may have noticed Link option under the Text menu. It's not one of the text styles that you would usually choose. In fact, specifying any text as Link results in it appearing much like the links you see in web browsers. Link style allows a programmer to override Revolution's definition of a word. All contiguous characters that have been linked are considered to be one word.

In this application we could use it to deal with "bound forms," i.e., multiple words that function as a semantic unit. Suppose *zapatos de tenis* were an idiomatic expression or slang term (like *tennie runners*). We can select the entire phrase and link the words. They now will be handled as a unit: they will highlight and be searched for in the glossary.

As mentioned before, linked text initially looks like links in a web browser, i.e., underlined and with a blue color. For our reading text it would look a little bizzare and visually disconcerting to have some words colored and underlined. Fortunately, Revolution provides a way to change the appearance of linked text without changing its function. In the **Colors & Patterns** section while inspecting the stack properties, we find that we can indicate the color of the three states of linked text. We need to change all of these to black (or to whatever color we decide the text to be). Under the **Basic Properties** for a stack we find the **Underline links** property. By deselecting that, none of the text is underlined. Now any text has a uniform appearance, but still functions as designed.

## Shakespeare Text

Now we'll look at a different kind of "Culture Reading" where it is assumed the reader already knows most of the words in the text and only needs help with unfamiliar terms. The example here is the famous witches scene from Shakespeare's *Macbeth*. Even native speakers of English need help with Shakespeare!

In a printed edition of Macbeth you would typically see difficult terms marked with superscipt numbers that cross-reference to notes at the bottom of the page or along the side. We would like to emulate this with a computer application. As you have probably surmised, this is similar to our previous problem, but in this case we want only certain words to be clickable. To facilitate this, we create a separate fields with a list of the unfamiliar terms which we wish to define (now this could be a separate stack or separate card, like the Spanish glossary). We'll put one word/phrase per line and we'll also create a parallel list of the definitions (again, one definition per line), making sure that they correspond to the wordlist (e.g., line 1 in the wordlist corresponds to line 1 in the definition list). If the **Don't wrap** property is set on each field, then it makes it easier to ensure that you have a line to line correspondence between the two fields. On the main text card we'll link the unfamiliar terms in the text itself (whether they are a phrase or a single word). We'll create a button named "Show Vocabulary" and script it to alternately show and hide links, to give users a visual cue as to which words have been defined.

We would now want appropriate mouse event handlers in the script of the text field to handler all requests for supplementary information:

```
on mouseDown
  if the textStyle of the clickText is link then
    put the clickText into whichWord
  else
    exit mouseDown
  end if
  find word whichWord in fld "Keys"
  get word 2 of the foundLine
  get line it of fld "meanings"
  put it into fld "defBox"
  get the clickLoc
  add 10 to item 2 of it
  set the left of field "defBox" to item 1 of it
  set the top of fld "defBox" to item 2 of it
  show fld "defBox"
  find empty
end mouseDown

on mouseUp
   hide fld "defBox"
end mouseUp

on mouseRelease
   hide fld "defBox"
end mouseRelease
```

First we need to check if the clickChunk is link style. If so, we'll put it into a variable; if not, we'll ignore it. We can thus effectively eliminate other words. We need to then go to the wordlist and find the word in a specific field. Due to the way we've arranged the words and their definitions, it will be necessary to use the foundLine function, which returns `line i of field 1`. The line number is all we need from this in order to get corresponding line from definition field (and consequently the proper definition). Having acquired the proper definition, we can put the definition into a field to display

Let's take advantage of the power of the computer and have the definition appear next to the word being defined. We employ a `mouseDown` handler to have the definition appear when the mouse is down and then have corresponding `mouseUp` and `mouseRelease` handlers to hide the field when the mouse is released. We need to position the field with the defintion relative to where the user clicked. We can get the clickLoc (it will be in `x,y` format) and add 10 to the vertical coordinate (the second item, or `y`, in the clickLoc). We can then use those coordinates to set the top and left of the field with the definition. Since we added 10 the vertical coordinate, the definition will appear below the word/phrase being defined.

## Another Approach

Let's create a hyperlinked set of scriptures by employing a method of bringing the user's attention to linked items within the text. This can be accomplished by highlighting the linked text as the user moves the mouse over it. We can also link other actions to this event, such as displaying the footnotes attached to that particular word or phrase. The controlling handler would be something similar to this:

```
local prevHilite

on mouseMove
  if prevHilite is not empty and prevHilite is not the mouseChunk then
    set the foregroundColor of prevHilite to "black"
    set the backgroundColor of prevHilite to empty
    put empty into prevHilite
    put empty into field "footnote"
  end if
  if the mouseChunk is not empty and "link" is in the textStyle of the mouseChunk then
    set the foregroundColor of the mouseChunk to "green"
    set the backgroundColor of the mouseChunk to "dark green"
    put the mouseChunk into prevHilite
    put the mouseText into thisRef
    put word 2 of the mouseLine into thisLine
    repeat forever
      find whole thisRef in field "wordList"
      if the result is "Not found" then exit mouseMove
      put word 2 of the foundLine into possLine
      put line possLine of field "wordList" into footnote
      set the itemDelimiter to "¥"
      if item 1 of footnote = thisLine then exit repeat
    end repeat
    put line possLine of field "footnotes" into field "footnote"
    find empty
  end if
  pass mouseMove
end mouseMove

on mouseLeave
  if prevHilite is not empty then
    set the foregroundColor of prevHilite to "black"
    set the backgroundColor of prevHilite to empty
    put empty into prevHilite
    put empty into field "footnote"
  end if
  pass mouseLeave
end mouseLeave
```

As with previous methods, this employs two data fields, one with the list of linked text and the other with the footnote data. We also need to distinguish between identical phrases with different associated footnotes. So each word/phrase in the word list field has a number associated with it identifying it with the verse to which it is attached. Again, the word list and the footnote fields correspond line to line.

The handler begins with a script local variable, meaning it is accessible to each handler within the script. This will contain the chunk data for the highlighted word or phrase. We intercept the `mouseMove` message to highlight the appropriate text with the user's movement of the mouse over the passage. First we check the contents of the variable to see if it contains chunk data and compare it to the present location of the mouse. If different, then the color attributes of the text are set back to normal, the variable is emptied, and the footnote field is cleared. This is the portion of the handler that un-highlights the text upon leaving it.

The next control structure is the poriton that highlights the text that are footnoted. It first checks to ensure that the user is indeed over a word and that the word is linked. Once those conditions are met, the color attributes of the text are changed to indicate its special status. We also store the chunk expression of that text in the special variable (to remove the highlight later as described above). We obtain the actual text and line number and start our search. It is set up such that the text searched for (`thisRef`) must also be associated with the particular verse (`thisLine`). The `repeat forever` loop will continue until both conditions are met (use with caution!). Once found, the correct footnote information is displayed. The `find empty` is necessary for the next footnote to be found correctly.

Technically the `mouseLeave` handler set up to remove the highlight is overkill and not necessary, as the first control structure in the previous handler should remove the highlight in all cases. However, you should always account for the possibility that the user may be quick and that the processor may be slow. This handler will "mop up" in the off chance the previous handler misses it.

## Conclusion

Once again, these methods are not necessarily the most polished approach to the task at hand. They are definitely not the only way to approach these tasks. For example, we have been dealing exclusively with text. There is the possibility of linking to pictures, sound, and video if one wishes. You have at your disposal many tools to achieve these same ends, and often in a more elegant and resourceful manner. Trust yourself and use your creativity.

---

## Now It's Your Turn

Your assignment, then, is to create a culture reading with hyperlinked text.

1. Create a stack and entitle it ***YourName*--CultRead**. Give the stack an appropriate label (based on the language/subject with which you are working).
2. Choose one of approaches outlined in these notes or one of your own liking where at least ten words are hyperlinked to supplementary information. If you decide to create a glossary, make sure that it accompanies your stack by putting them both within a folder (appropriately named, of course).
3. Put a copy of your finished stack (or folder with stacks) in the **Assignments** folder.

Course Schedule
Main Page