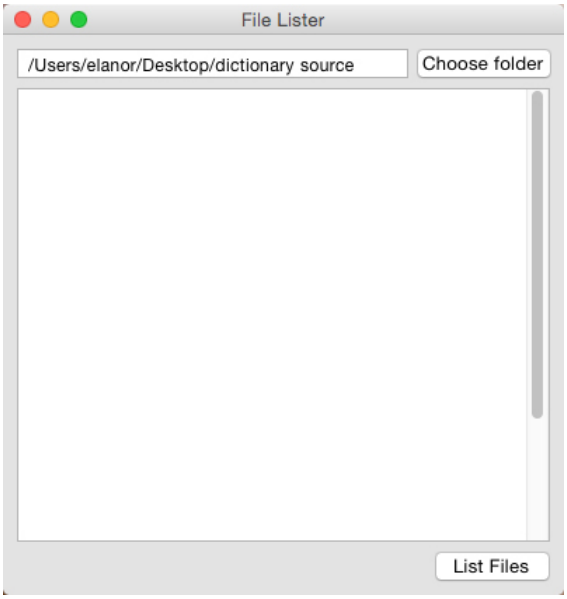
Files and Folders Part 2

In Part 1 of this lesson we looked at how to list files and folders in LiveCode. In this lesson we will extend on what we have already learned by using recursion. Recursion is simply the name given to the process where a handler or function calls itself.

Recursion can allow problems to be expressed very succinctly and when properly used, it can make your code shorter and easier to read. However, recursion is often less efficient than iteration (using loops instead) and can lead to cryptic code that you have difficulty understanding yourself. If, when using recursion you find that you are struggling to understand how your code works, it is probably a good idea to try simplifying the code by using loops, even if this makes the scripts longer.

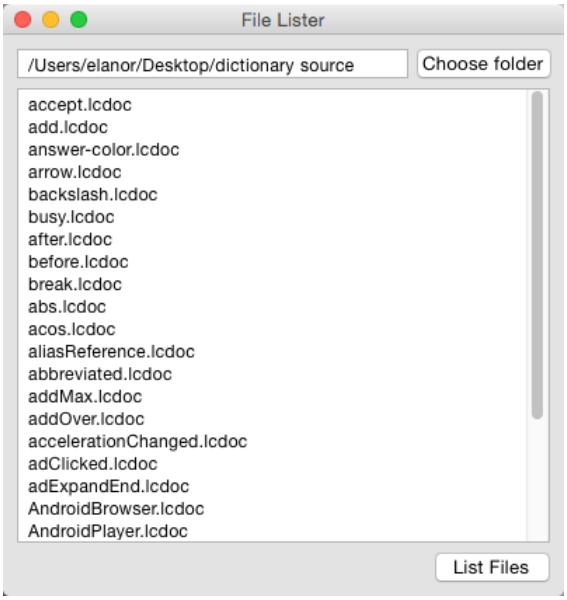
Using Recursion to list the files in a folder



Often simply getting the list of files in a single folder is not enough, many applications need also to include the files in sub folders. In English you can say that the list of files in any given folder consists of the list of files in that folder plus the list of files in each sub folder, this is exactly how this algorithm works, first listing the current files, then adding the list of files for each sub folder.

We will start by setting up our interface as we did in Part 1. A field and button allowing us to choose a folder, a field to display the list of files and a button to get the list of files.

The Recursive Algorithm



As mentioned above a recursive algorithm is one that calls itself, be careful to include an exit case or your program could end up in an infinite loop.

For our task we have a function listFiles which gets the contents of a folder, lists the files, and calls **listFiles** on each subfolder. We use a variable to store the list of files, adding to it each time we call **listFiles**. In this function we want to stop recursing when there are no more subfolders.

```
# Returns a list of files in a given folder, using recursion to
# include files in subfolders if desired.
function listFiles pFolder, pRecurse
	local tTotalFiles, tCurrentFiles, tFolders

	set the defaultFolder to pFolder
	put filteredFiles() into tCurrentFiles

	if not pRecurse then
		return tCurrentFiles
	end if

	if tCurrentFiles is not empty then
		put tCurrentFiles & return after tTotalFiles
	end if

	## Now the recursion
	## We call listFiles passing a subfolder as an argument
	put filteredFolders() into tFolders
	repeat for each line tFolder in tFolders
		put listFiles((pFolder & slash & tFolder), pRecurse) into tCurrentFiles
		if tCurrentFiles is not empty then
			put tCurrentFiles & return after tTotalFiles
		end if
	end repeat
	delete the last char of tTotalFiles

	return tTotalFiles
end listFiles
```

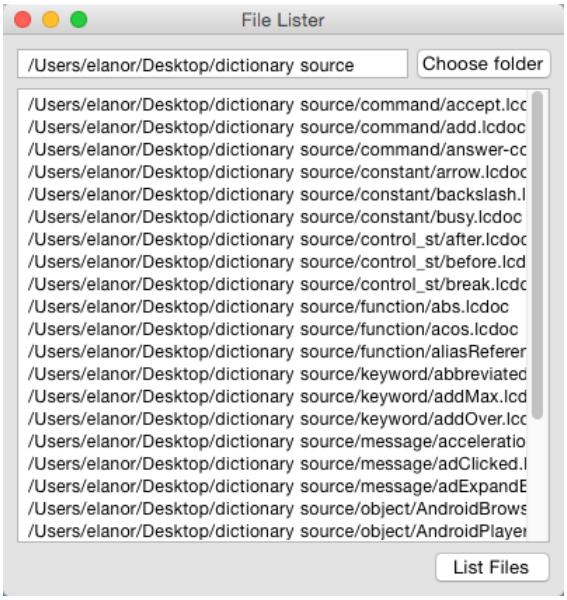
The filtering functions are the same as those in Part 1 of this lesson.

Hints on using recursion in LiveCode

Hint: Note the way that repeat for each is used to loop through each folder. This is the most efficient way in LiveCode of looping through a list, and is much faster than using repeat with x=1 to the number of lines of tFolders. Remember to delete the last character of the created list after the repeat has finished!

Hint: It is very easy to change this function to list folders instead of files, or to list both folders and files. To list folders instead just change filteredFiles() for filteredFolders() and be sure to rename the function and local variables so that they make sense. To list folders you simply add the filteredFolders() onto the end of the filteredFiles().

Using Recursion to list the full paths to the files in a folder



We can also use a recursive function to give us the full paths to all the files in a folder. Again the **filteredFilesWithPaths** function is the same one we used in Part 1, it simply prepends the defaultFolder to the filename of each file to give the full path.

```
# Returns a list of files with the full paths
function listFilesWithPaths pFolder, pRecurse
	local tTotalFiles, tCurrentFiles, tFolders

	set the defaultFolder to pFolder
	put filteredFilesWithPaths() into tCurrentFiles

	if not pRecurse then
		return tCurrentFiles
	end if
	if tCurrentFiles is not empty then
		put tCurrentFiles & return after tTotalFiles#
	end if

	## Now the recursion
	## We call listFilesWithPaths passing a subfolder as an argument
	put filteredFolders() into tFolders
	repeat for each line tFolder in tFolders
		put listFilesWithPaths((pFolder & slash & tFolder), pRecurse) into tCurrentFiles
		if tCurrentFiles is not empty then
			put tCurrentFiles & return after tTotalFiles
		end if
	end repeat
	delete the last char of tTotalFiles

	return tTotalFiles
end listFilesWithPaths
```

Listing Sorted Files

The return values of the files and folders functions in LiveCode are already sorted alphabetically, but if you want to obtain a properly sorted list of of files and or folders, it is necessary to sort the complete list after all recursion is complete. A flexible way of doing this is shown below.

```
# Returns a sorted list of the files in pFolder. Again using
# recursion if required.
function listSortedFiles pFolder, pRecurse
	local tFiles

	put listFiles(pFolder, pRecurse) into tFiles
	sort lines of tFiles by listSortedFilesSortKey(each)

	return tFiles
end listSortedFiles

# Used by the listSortedFiles() function. Returns a sort key
# from each file's name to allow the sorting to be fully customized.
function listSortedFilesSortKey pFile
	# Use this value for a normal text sort
	return pFile
end listSortedFilesSortKey
```

This code works by iterating through the list of files, and for each file, calling the listSortedFilesSortKey() function. This function takes the file's name and returns the string that LiveCode should use to sort this file. For a standard alphabetical sort, the name of the file is returned, but there are many other options, for example you could sort on the file's complete path instead by simply changing return pFile to return the defaultFolder & slash & pFile. Other possibilities include inverse sort, numeric sort and sort by extension which can be easily implemented.

1 Comments

Richarc Gaskin Wednesday Jan 22 2020 at 01:09 PM

There is a bug here that will cause recursion: when obtaining "the files", we have to check the result for errors. Without that, attempts to drill down into them will keep accessing the same inaccessible folder, giving rise to a recursion error.

Also, the function that obtains that file/folder list does not appear to be defined in this code listing.