Computers & the Humanities 281

Working with External Files

When dealing with any type of application development, particularly on a large scale, it often becomes necessary to work with data contained in files other than the current stack. Doing so effectively and efficiently is a challenge, but it also increases the elegance with which your application executes itself.

The primary reason for working with external files is to decrease the actual memory required by your application. If the data for your resources are contained in discrete files external to the actual application, then the application itself will run more quickly and smoothly as it will not be bogged down by an inordinate amount of data. Another reason for having external files is to allow access to such files by multiple applications. These will each be covered in turn.

Referencing External Files

In order to work successfully with external files, it is first essential to have a correct understanding of file paths, which are basically descriptions of the exact location of a particular file. Such paths are created by beginning at the top of the computer's file system (which usually happens to be the disk drive's name), and then moving down the file structure, naming every folder that encloses the file, in descending order, until the file is reached.

For example, `/Hard Disk/Tutorials/German/101/Schmetterling.gif` is a file path pointing to an image file (Schmetterling.gif) located within a folder entitled "101" which is located in a folder called "German" located in a folder called "Tutorials" which resides on the disk drive entitled "Hard Disk." In this particular case, this path is an absolute path pointing to an image.

Revolution possesses a few properties that allow us to access and manipulate and work with relative paths (as opposed to absolute paths). The `defaultFolder` is a property of the currently open stack or application. It is a string that gives the path to the folder containing the application. In the authoring environment, this is path points to the folder containing Revolution. If you have created a standalone, then it points to the folder creating a standalone. This path can be obtained commands such as:

```
get the defaultFolder
put the defaultFolder into mainPath
```

It is also possible to set this property to an arbitrary value chosen at the discretion of the developer or the user:

```
set the defaultFolder to "/Hard Disk/Tutorials/Humanities101/Picts"
```

Once the value of the defaultFolder is known (whether accessed or set), it may be used or manipulated to create access to outside resources. Folder names can be added or subtracted to the path according to need. This is especially useful when you are not sure of the precise location where your application will be deployed. The use of relative paths allows greater flexibility and minimizes the possibilty of broken resource links that happen with absolute paths.

The importance of the value of the defaultFolder is underscored by the fact that files of all types located within this folder can be accessed or referred to within the application without utilizing the entire path. The name of the file itself is sufficient. The details for this particular process will be explained later in the context of the various objects with which we will be working.

Images

As mentioned before, image data can reside in the stack or outside the stack in an external file. A small number of images requiring small amounts of data are acceptable as parts of the stack. However, if the need arises for a large number of images with higher resolution, you may be better off by placing such in an appropriately named resource folder created in conjunction with your stack.

If we are working in a fixed environment and know that the resources will be located in an absolute location (such as on a remote server, or on the particular hard drive), then we can establish our images with absolute paths (it would be best to have all the picture files in the same folder, for the sake of organization). With this setup, the application can be placed in different locations, and as long as the path to the picture files is accessible, the images will be displayed properly.

If we were to create a standalone and were not sure where it would be deployed, absolute paths would be useless. In this situation it would be best to place all picture files in a single folder and bundle those with the application. Instead of creating an absolute path, we would need to set the link of each image to something similar to `./Picts/Schmetterling.jpg` where "Picts" is the folder located in the default folder. Using "./" indicates that you wish the application to use whatever exists or has been set as the current folder.

Audio and Video

The process for accessing audio and video files as external files is so similar for each type that we will treat them together. The access to these files is achieved through the use of player objects. Players are control objects just like buttons and fields. Consequently they share some common properties (such as name, visible, location, etc.), as well as possessing some properties unique to their object type.

As with images, if we know the application will be deployed in a static environment, then we can create the links to the audio and video files as absolute paths. Again, it would be best to organize the media into appropriate folders.

Creating relative paths would be similar. If the files are all located within the same folder as the application, then referring to the files by their name alone will suffice. However, if they are in another folder, we would have to set the defaultFolder to an appropriate value, or we could set the link for each file to an appropriate value, such as `./Videos/Bonjour.mov` or `./Sounds/AuRevoir.wav` where "Videos" and "Sounds" are folders within the default folder.

Text

Revolution has the capability of reading and writing to external text files. The easiest way to accomplish this is through the `URL` keyword. The other necessary keyword is `file`. Some examples of proper syntax for this would be:

```
get URL "file:results.txt"
put URL "file:/Hard Disk/Documents/DatesTimes" into field "whichTime"
put the result into URL "file:./Datafiles/feedback.txt
put field "finalAns" after URL "file:/Hard Disk/Tutorials/Datafiles/Results"
```

As with other external files, you can either write out the entire path to the location for the text tile, or you can create a relative path as indicated earlier for other object types. With that path you can either put the contents of the file into another container, or you can write the contents of a container to a text file. The keywords `into`, `before`, and `after` work as they do with putting text into other containers.

Conclusion

The key to working with external files is to ensure that you have a correct knowledge of file paths and use them appropriately as the conditions require.

Course Schedule
Main Page