Handy Handlers #6: EnsurePath

"As we know,
There are known knowns.
There are things we know we know.
We also know
There are known unknowns.
That is to say
We know there are some things
We do not know.
But there are also unknown unknowns,
The ones we don't know
We don't know."
- Donald Rumsfeld

When writing to a data file using the "open" or "put" commands, LiveCode will create the file for you if it doesn't already exist. But what about the folder the file will be created in? To ensue that the folder will be there when you need it, just run the path though this function and it'll take care of the rest.

Using the "there is" and "there is not" operators
LiveCode provides a built-in operator for checking the existence of just about anything, from controls and stacks to files and folders. In keeping with Revolution's English-like flavor, this operator is simply called "there is", and using it is a snap:

```
if there is a stack "MyPefs" then open stack "MyPrefs"

if there is not a player 1 then create player

if there is a not file "MyDrive/MyApp/Data/MyData.txt"
then
     put tMyData in url "file:/MyDrive/MyApp/Data/
MyData.txt"
end if
```

That last example looks simple enough, but like most things in programming it's the error-checking that will make the difference between code that works once for the programmer and code that's

worthy of deployment to end-users. Specifically, that example doesn't check if the folder "/Data/" exists.
We can modify that snippet to check for the folder first easily enough:

```
if there is not a folder "MyDrive/MyApp/Data/" then
  create folder "MyDrive/MyApp/Data/"
end if
if there is not a file "MyDrive/MyApp/Data/MyData.txt"
then
     put tMyData into url "file:/MyDrive/MyApp/Data/
MyData.txt"
end if
```

That's a start, but what if the OS can't create the folder, such as if we tried to create it in a locked volume or one we don't have permission to modify?

Adding Error-Checking
We'll add an error-check after the "create folder" command, checking the value of "the result" which will contain an error message if there was a problem which prevented the folder from being created, or will be empty if not:

```
if there is not a folder "MyDrive/MyApp/Data/" then
  create folder "MyDrive/MyApp/Data/"
  if the result is not empty then
    answer "Couldn't create folder."
    exit to top
  end if
end if
if there is not a file "MyDrive/MyApp/Data/MyData.txt"
then
     put tMyData into url "file:MyDrive/MyApp/
Data.MyData.txt"
end if
```

We're further along than we were, providing notice to the user if the folder couldn't be created. But we haven't told the user which folder couldn't be created, and if they need to call tech support â€“ that may mean you â€“ you'll want to know that too.
So we'll append our error-check to include the folder path:

```
if there is not a folder "MyDrive/MyApp/Data/" then
  create folder "MyDrive/MyApp/Data"
  if the result is not empty then
    answer "Couldn't create folder: MyDrive/MyApp/Data"
    exit to top
  end if
end if
if there is not a file "MyDrive/MyApp/Data/MyData.txt"
then
     put tMyData into url "file:MyDrive/MyApp/
Data.MyData.txt"
end if
```

Now we're off to a good start, and can create our file knowing the folder we want to create it in will be there. But what if we want to change that folder later on, or use a similar error-check in some other part of our program that writes files?
We could copy and paste the code each time we use it, but that's not merely inefficient, more importantly it isn't fun. Let's turn that into its own separate function instead, so we can call it from anywhere.

Turning the Error-Checking into a Separate Function
Of course the first thing we have to do is decide on a name for this function. Since it ensures that a folder path will be available, we can call it "fwEnsurePath".
The "fw" prefix helps you remember where you got this handler from, and better ensures that it'll avoid conflict with any new tokens added to the Revolution engine in the future, or functions in other libraries you might use. For more on this and other naming conventions see "Scripting Style Guide".
This function will need only one argument, the path we're ensuring, so we'll be able to call it like this:

```
 get fwEnsurePath("MyDrive/MyApp/Data/")
```

This gives us a one-line way to ensure that any path we want to write to will be there.
Having defined how we want to use it, we can now get down to writing the function itself.
Since we want it to be able to handle any path, which may refer to any number of subfolders, we'll need to walk through each part of the path

one directory at a time, checking each folder along the way and creating it as needed, or reporting an error if the folder couldn't be created.
To handle any number of folders in the path we'll use a repeat loop, and we can parse the string of paths easily if we first set the itemdelimiter to the character LiveCode uses to delimit folders in paths, "/". Within the repeat loop we'll build a copy of the path in a variable named tTestPath, adding a new folder leaf and checking it each time through the loop.
By the time we're done each folder will have been checked, and if no error was reported the path passed to it will be ensured.
Here's how these elements come together in the completed function:

```
   function fwEnsurePath pPath
     set the itemdel to "/"
     --
     put item 1 of pPath into tTestPath
     put pPath into tDestFolders
     delete item 1 of tDestFolders
     repeat for each item tFolder in tDestFolders
        put "/"&tFolder after tTestPath
        if there is not a folder tTestPath then
           create folder tTestPath
           Err the result, "Couldn't create folder
"&quote&tTestPath&quote
        end if
     end repeat
     return pPath
   end fwEnsurePath
```

Note that instead of using the answer command to report any error to the user, instead we used a call to an Err command, which we defined in Handy Handlers #1.
As the series progresses you'll find more of the code in this articles making similar use of commands and functions we've already written. For example, in the next article you'll see how we put fwEnsurePath to work in creating a simple pair of handlers for writing and reading preferences files.