Handy Handlers #2: GetFileData - easy file I/O across platforms

"Increased reuse of code and design is often cited as a major reason for adopting a new programming language or design strategy. However, most organizations reward individuals and groups that choose to re-invent the wheel."
- Bjarne Stroustrup

Reading a file's contents is something nearly all programs do, and Transcript makes it easy with something like this:

```
answer file "Select a file:"
if it is empty then exit to top
put url ("file:"& it) into fld 1
```

The only downside to that particular code is that the call to the "answer file" command lets the user select any file, but you may want to handle only a particular type. For example, if you were making a text viewer you'd want to help the user avoid accidentally selecting an image file.
All modern operating systems provide a standard "Get File" dialog for selecting a file which includes a means of filtering the range of selectable file types. LiveCode supports these filters well, but they work differently for each platform.
On Windows and Unix systems, a file's type is determined by its file name extension, a set of one to four characters following the last period in the file name. Common file type extensions include ".txt" for text files, ".mov" for QuickTime media, and of course ".rev" for LiveCode stacks.
To filter the operating system's Get File dialog on Windows and Unix systems you use the "with filter" option like this:

```
answer file "Select a text file:" with filter "*.txt"
```

Macintosh is a different story. While file extensions play a role in determining a file's type under OS X, in older versions of Mac OS the type is determined by hidden Finder info, with the file type extension coming into play in OS X only in cases where this hidden Finder info is not present.
To filter files on Mac OS you use the "of type" option, like this:

```
answer file "Select a text file:" of type "TEXT"
```

So if you want to filter file types on all platforms in one handler you'll need to query the "platform" function and use the appropriate form for the OS your user is running:

```
if the platform is "MacOS" then
```

```
  answer file "Select a text file:" of type "TEXT"
else
  answer file "Select a text file:" with filter "*.txt"
end if
if it is empty then exit to top
put url ("file:"& it) into tData
```
Notice that I also added some simple error-checking. The second-to-last line now checks to see if the value returned in the local variable "it" is empty, which would be the case if the user selected the Get File dialog's "Cancel" button; if so we exit script execution.
As you saw in the previous installment in this series, I'm too lazy to type any more than necessary, and I encourage such laziness in others. With the snippet above we can read any text file on any platform, but it means writing seven lines of code each time we do. If we break it out into a separate handler, however, we just tuck it in the script of the mainstack or a library, and whenever we need to read a file we can just write one line:
```
on mouseUp
  put GetFileData() into fld 1
end mouseUp
```
...with this in our mainstack script:
```
function GetFileData
  if the platform is "MacOS" then
    answer file "Select a text file:" of type "TEXT"
  else
    answer file "Select a text file:" with filter
"*.txt"
  end if
  if it is empty then exit to top
  return url ("file:"& it)
end GetFileData
```
Now it's time to add some features. First, the function has only one prompt string to direct the user, and we may want to give different directions depending on where in our program we're using it.
Another problem is that our function filters for just one file type, and we may want to use this for reading all sorts of files, perhaps even file types unique to our application.
To fix the first problem we'll add an argument to the function called pPrompt, which will allow us to pass in a string to display in the Get File

dialog. And since we're lazy we'll make it optional, so if we're just throwing something together quickly we can call this function with no arguments and still have a useful string displayed in the dialog:

```
function GetFileData pPrompt
  if pPrompt is empty then put "Select a file:" into
pPrompt
  if the platform is "MacOS" then
    answer file pPrompt of type "TEXT"
  else
    answer file pPrompt with filter "*.txt"
  end if
  if it is empty then exit to top
  return url ("file:"&it)
end GetFileData
```

Now we need to modify it to let us specify a file type. Given the differences between Mac OS file type codes and Windows and Unix file name extensions, we'll add two arguments, one for Mac and one for the other OSes:

```
function GetFileData pPrompt, pMacType, pWinType
  if pPrompt is empty then put "Select a file:" into
pPrompt
  if the platform is "MacOS" then
    answer file pPrompt of type pMacType
  else
    answer file pPrompt with filter pWinType
  end if
  if it is empty then exit to top
  return url ("file:"&it)
end GetFileData
```

Now we can call this anywhere, specifying our prompt and file types:

```
on mouseUp
  put GetFileData("Select a text file:", "TEXT",
"*.txt") into fld 1
end mouseUp
```

By now you may be asking, "But what if I want to read a binary file?", and since LiveCode provides support for text and binary file I/O, there's no reason not to.

Consider the following:

```
put url ("file:"& it)
```
The "file:" part of the local URL tells LiveCode to use its text-reading mode, which modifies any line-endings in the file to convert them to the Unix standard line-ending used internally, ASCII 10 (in LiveCode denoted with the constant "return" or its abreviated form "cr"). This is especially useful for making multi-platform applications which display the contents of text files, since you can write them on any platform and display them on any other platform and the line endings will display correctly.
Reading a file in binary mode does not modify the data as it's read, giving, you exact copy of what's in the file. To read a binary file using the URL form we just use "binfile" in place of "file":
```
answer file "Select a file:"
if it is empty then exit to top
put url ("file:"& it) into fld 1
```
1
To support binary mode as an option we'll add one last argument to our function called pTextOrBinary, and we'll check it by adding this:
```
if pTextOrBinary is in "binary" then
  put url ("binfile:&it) into tData
else put url ("file:&it) into tData
```
Note that the checking is done by determining if the value of pTextOrBinary is in the string "binary". This lets us use "binary" or even just simply "bin" and the result is the same, and we can use the word "text" or leave it out entirely if we want to read the file using text mode.
We're almost done, but our function has one critical weakness: poor error-checking. We do check whether the user cancelled the Get File dialog, but we still need to check for file I/O errors after we attempt to read the file with the "put url" command. In computing, every day is Anything Can Happen Day, so you can almost never be too careful about error-checking.
Since we already wrote a generic error-handler last week, all we need to add is one small line with our Err handler to complete our function:
```
function GetFileData pPrompt, pMacType, pWinType,
pTextOrBinary
  if pPrompt is empty then put "Select a file:" into
pPrompt
  if the platform is "MacOS" then
    answer file pPrompt of type pMacType
  else
```

```
    answer file pPrompt with filter pWinType
  end if
  if it is empty then exit to top
  if pTextOrBinary is in "binary" then
    put url ("binfile:"&it) into tData
  else put url ("file:"&it) into tData
  Err the result
  return tData
end GetFileData
```

Now that we can let the user select a file and read its contents on any platform, next week we'll break this function into two functions for more than twice the functionality.