# Handy Handlers #3: GetFileData revisted

*"You know you've achieved perfection in design,Â*
*not when you have nothing more to add,*
*but when you have nothing more to take away.*
*-Antoine de Saint Exupery*

In the last installment of this series we explored a generic handler for reading files, and after adding a few features the final version was:

```
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
```

Remember that this function uses the Err handler we created in the first edition of this column.

This week we're going to throw it in the trash and start over. More specifically, we'll break it into two separate functions to get more than twice the flexibility.

What we have in that function is really two separate tasks: getting a file path, and then using that to read the data from the file.

But what if we want just the path to a file without reading its data? There are many good reasons to do so, like letting the user pick a QuickTime file to present in a player object, or to set the filename property of an image.

The simplest way to break up this function might be like this:

```
function GetFileName 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 it
end GetFileName

function GetFileData pFileName, pTextOrBinary
  if pTextOrBinary is in "binary" then
    put url ("binfile:"& pFileName) into tData
  else put url ("file:"& pFileName) into tData
  Err the result
  return tData
end GetFileData
```

We can call these in sequence like this:

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

But now we have to use two lines to get the functionality we had previously enjoyed with just one. Let's see what we can do to bring our calling script back to a simple one-liner.

The challenge of course is that the parameters needed by each function are different, so we'll have to add some to one so it can pass them to the other.

Let's change the GetFileData function so that the first argument, the file name, can optionally be the word "ask" or just an empty string to act as a flag that it should call GetFileName to get a path to a file:

```
function GetFileData pFileName, pTextOrBinary, pPrompt,
pMacType, pWinType
  if (pFileName = "ask") OR (pFileName = "") then
    if pPrompt is empty then put "Select a file:" into
pPrompt
```

```
    put GetFileName(pPrompt, pMacType, pWinType,
pTextOrBinary) \
      into pFileName
  end if
  if pTextOrBinary is in "binary" then
    put url ("binfile:" &pFileName) into tData
  else put url ("file:" &pFileName) into tData
  Err the result
  return tData
end GetFileData
```

Now we have one function that lets us get a file's data in one line, and while it offers many options, calling it can be as simple as:

```
put GetFileData("ask") into fld 1
```

...or even just:

```
put GetFileData()
```

And with these functions separated we now have a good toolkit for a wide variety of file-related tasks.

To assign a QuickTime movie for playing in a player object:

```
set the filename of player 1 to \
  GetFileName("Select a movie:", "MooV","mov")
```

To store the contents of a binary file in a custom property:

```
set the uMyProp of this cd to \
  GetFileData("ask", "binary")
```

To put a specific text file into a field:

```
put GetFileData("MyFolder/MyFile.txt") into fld 1
```

To let the user select a text file to display in a field:

```
put GetFileData("ask", "text", "Select a text file:",
"TEXT", "txt") \
  into fld 1
```

To let the user select a JPEG file for display in an image object:

```
set the fileName of img 1 to \
GetFileName("Select a JPEG image:",
"JPEG","*jpeg;*jpg")
```

And we get all this easy-to-use functionality by just putting these two functions in our stack script:

```
function GetFileName 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 it
end GetFileName

function GetFileData pFileName, pTextOrBinary, pPrompt,
pMacType, pWinType
  if (pFileName = "ask") OR (pFileName = "") then
    if pPrompt is empty then put "Select a file:" into
pPrompt
    put GetFileName(pPrompt, pMacType, pWinType,
pTextOrBinary) \
      into pFileName
  end if
  if pTextOrBinary is in "binary" then
    put url ("binfile:&pFileName) into tData
  else put url ("file:&pFileName) into tData
  Err the result
  return tData
end GetFileData
```

It took us a bit of work to revise the functions to be this flexible, but with the convenience they provide it was well worth the effort.

Next week we'll make a handler to replace one line from these. Why replace a one-line statement with a one-line handler call? Tune in next week and find out. While we're at it we'll take a look at how to use naming conventions for variables and parameters to make your code more readable and maintainable.