Handy Handlers #1: Err - a generic error handler

*The first in a series of new **LiveCodeJournal** columns by Fourth World's Richard Gaskin. This week, Richard tells us why to Err is divine...*

"God is in the details."
- Ludwig Mies van der Rohe

Welcome to the first Handy Handlers column where each week we'll explore a useful handler you can use in your own scripts, showing you how it's put together and how to use it.

We kick off the series with a simple generic error handler I call Err. There are many options for handling errors in Transcript, but I find myself using Err most often because it's simple to use and covers the basics.

In LiveCode, the result function returns the status of the last find, go, open, send, or file operation. A common way to handle errors is to check the result and if it's not empty present it to the user, usually terminating execution afterward:

```
open file "MyFile"
if the result is not empty then
  answer the result
  exit to top
end if
```

Writing those four error-handling lines each time you need them may not be a big deal in a small project, but in a larger one it can be cumbersome. More importantly, if you want to change the way the error is handled later, such as logging it to a file, you'd have to go back through your code any modify every error-handling section.

By breaking that error-handling out into a separate handler you'll reduce the amount of code you type and have a system that's far easier to maintain.

We could rewrite the example above as:

```
open file "MyFile"
Err the result
```

In your mainstack script of wherever else you put handlers to be accessed by objects throughout your program, you include this:

```
on Err pResultString
  if pResultString is not empty then
    answer pResultString
```

```
    exit to top
  end if
end Err
```
That's handy, but what if we want to present a more complete message for the user than LiveCode's sparse error message text? The phrase "can't open file" is useful for us programmers but not especially helpful to our app's users. Which file? Why can't it be opened?
We can provide a more complete message as an optional argument to the Err handler:
```
open file "MyFile"
Err the result, "The file MyFile could not be opened."
&cr&\
  "Is it in use by another application?"
```
Now we modify our mainstack's Err handler to support this addition:
```
on Err pResultString, pDisplayMessage
  if pResultString is not empty then
    if pDisplayMessage is empty then
      put pResultString into pDisplayMessage
    end if
    answer pDisplayMessage
    exit to top
  end if
end Err
```
Note the section:
```
if pDisplayMessage is empty then
  put pResultString into pDisplayMessage
end if
```
By first checking if the pDisplayMessage argument was included we make it optional. You can provide a display string if needed, or keep it simple and leave it out if the LiveCode result will suffice.
There's still one more modification to make this handler useful in a wider range of circumstances.
As shown above, the Err handler always exits all subsequent execution and returns to idle if the result value passed to is not empty. In most cases that's exactly what we want, as there'd be no point in trying to process a file if it can't be opened.
But there may be times when you want to notify the user of an exception and then continue. For example, if a particular file is not available you might want to use another. So we'll add an optional third argument as a

flag to determine whether to exit to top or not, so the final version of our Err handler looks like this:

```
on Err pResultString, pDisplayMessage, pDontExitFlag
  if pResultString is not empty then
    if pDisplayMessage is empty then
      put pResultString into pDisplayMessage
    end if
    answer pDisplayMessage
    if pDontExitFlag is empty then exit to top
  end if
end Err
```

Note that we didn't check if our pExitFlag is true or false, but merely if it's empty. If we required either "true" or "false" we'd have to remember what that refers to whenever we review code that calls it. But as written above we can simply not include it when we don't need it and call it anything descriptive when we do:

```
open file "MyFile"
Err the result, "The file MyFile could not be opened."
&cr&\
   "Your backup file will be used instead.", "DontExit"
```

Now that we have a simple way to handle errors, next week we'll put it to use with a platform-independent way to get the contents of a selected file.

About the Author

Richard Gaskin is Ambassador of Fourth World Systems, a Los Angeles-based consultancy specializing in multi-platform software development and LiveCode training. With 15 years' experience, Richard has delivered dozens of applications for small businesses and Fortune 500 companies on Mac OS, Windows, Linux, and the World Wide Web.
http://www.fourthworld.com