Sons of Thunder Software
SOFTWARE DEVELOPMENT AND CONSULTING SERVICES

Products Services Developer Resources Contact STS About STS

livecode

The Beauty of "Pass By Reference"

Most of us who have migrated to MetaCard or Revolution from other scripting languages like HyperCard or SuperCard are familiar with the normal method of passing parameters to a handler or function... something like this:

```
on mouseUp
  -- imagine you had a login screen that took people's names, and you
  -- wanted to create a list of people who logged in using the global
  -- gUserList
  global gUserList
  ask "What's your name:"
  if it <> "" then
    put append(it,gUserList) into gUserList
  end if
end mouseUp

function append pName,pList
  put pName into line (the number of lines of pList)+1 of pList
  return pList
end append
```

In this case you pass in a name, and append it to a growing list of users, but you have to pass in both the name that was entered an the global to the function and store the returned value in the global. (True, you could add to the global *inside* the `append` function, but then it goes from being a general purpose function to a special purpose one, and we can't have that, can we?) This method is called **Pass By Value**; basically, you're passing the value of the variables to the function you wish to use.

Suppose, however, that you wanted to return the number of lines in the list as part of the `append` function so you can monitor the list and make sure it doesn't go over 100 users. In the model we're familiar with, you'd have to count the lines after you called the `append` function, like:

```
on mouseUp
  global gUserList
  ask "What's your name:"
  if it <> "" then
    put append(it,gUserList) into gUserList
    if the number of lines of gUserList = 100 then
      answer "OK, no more names can be added."
    end if
  end if
end mouseUp

function append pName,pList
  put pName into line (the number of lines of pList)+1 of pList
  return pList
end append
```

Both MetaCard and Revolution support a really wonderful way to handle these kinds of situations by a method called **Pass By Reference**, in which the *variable itself* is passed to the function, and any changes to its proxy are made to that variable. What that means is that you don't have to do anything with the returned value, because the function itself has updated your variable for you. The magic is in the **@** symbol which precedes a parameter to a function or handler; this indicates that the parameter is being passed by *reference* and not by *value*.

Here's the same original (non-counting) handler using **pass by reference**:

```
on mouseUp
  global gUserList
  ask "What's your name:"
  if it <> "" then
    get append(it,gUserList)
  end if
end mouseUp

function append pName,@pList
  put pName into line (the number of lines of pList)+1 of pList
end append
```

Note that the `append` function in this case doesn't have to return anything. In fact, it could have been turned into a simple handler:

```
on mouseUp
  global gUserList
  ask "What's your name:"
  if it <> "" then
    append it,gUserList
  end if
end mouseUp
```

```
on append pName,@pList
  put pName into line (the number of lines of pList)+1 of pList
end append
```

What's happening here is that `gUserList` is being passed into the `append` handler, and its *proxy*, `pList` gets updated by the handler, which actually updates `gUserList`. Cool, huh?

Now back to our counting situation... because you can update the value of a variable by passing it by reference to a handler or function, that means that the returned value of a function can be used for other things. Suppose we changed our `append` function to return the count of the number of lines in the variable being updated. This would allow us to do something like this (the original couting handler, updated for pass by reference):

```
on mouseUp
  global gUserList
  ask "What's your name:"
  if it <> "" then
    if append(it,gUserList) = 100 then
      answer "OK, no more names can be added."
    end if
  end if
end mouseUp

function append pName,@pList
  put pName into line (the number of lines of pList)+1 of pList
  return (the number of lines of pList)
end append
```

This will cause `gUserList` to be updated, **and** we get a count back that we can check.

As you can see, pass by reference parameters are really useful, and can cut down on code quite dramatically. Try them for yourself, and have fun!

Posted 1/18/2003 by Ken Ray

Print this tip

News and Rumors Products Services Developer Resources Contact STS About STS