[7.4] Return error codes as strings

It is good programming practice, and it will save you and your user lots of grief, if you write your programs to return an error code if the program cannot run normally. As a simple example, consider a function that finds the reciprocal of the argument:

```
recip(x)
func
return 1/x
endfunc
```

If x=0, this function will return *undef*, which doesn't tell the user what went wrong. Also, if this function is called deep within a series of function calls, it can be difficult to determine just what caused the error.

This is a better method:

```
recip(x)
func
if x=0 then
 return "bad arg in recip"
else
 return 1/x
endif
endfunc
```

Now the program checks for a valid argument before attempting the calculation. If the calculation can't be done, *recip()* returns a string instead of a number. The calling routine can use *GetType()* on the result to determine if an error occurred, and handle it appropriately. The returned string can be used as the error message displayed to the user, which tells him what went wrong (bad argument) and where it went wrong (recip).

This method fails if the function is designed to return a string instead of a number. In that case you may be able to use special coded strings for the error message. For example, suppose we have a routine that is supposed to build a string from two arguments:

```
stringit(a,b)
func
if dim(a)=0 or dim(b)=0 then
 return "¨bad arg in stringit"
else
 return a&b
endif
endfunc
```

stringit() is supposed to return *a* concatenated with *b*, but only if both arguments have one or more characters. If this is not true, then *stringit()* returns an error message, the first character of which is the 89/92+ comment symbol, character 169. In this case, the calling routine checks the first character of the result to see if it is character 169. If so, the error is handled.

This method assumes that the strings will not start with character 169.