Error Trapping and Resolution Patterns

These two sample patterns are presented as starting points for a discussion on useful error behavior patterns. If you have any thoughts, please share them. Both scripts find a record based on a specified primary key, and optionally find the record in the context of a specific layout.

Guard Clauses

Many sources suggest that error handling code be written as close as possible to the source of the error being handled. Guard clauses are often used to avoid the arrow anti-pattern in code structure, which can easily get out of control when accounting for the possibility of several different errors.

```
# Script: Find Record ( id ; { layoutName } )
Set Variable [$ignoreMe; Value:#AssignScriptParameters]
#
# Check that required parameters are populated
If [IsEmpty ( $id )]
	Exit Script [Result: # ( "error" ; 5 ) //Command is invalid]
End If
#
# Go to specified layout
Go to Layout [$layoutName]
If [Get ( LastError ) = 105 //Layout is missing]
	Show Custom Dialog ["The layout " & Quote ( $layoutName ) & " is missing."]
	Exit Script [Result: # ( "error" ; 105 )]
End If
#
# Find specified record
Enter Find Mode []
Set Field By Name [GetFieldName ( GetField ( "id" ) ); $id]
Perform Find []
If [Get ( LastError ) = 401 //No records match the request]
	Go to Layout [original layout]
	Show Custom Dialog ["The record " & Quote ( $id ) & " could not be found."]
	Exit Script [Result: # ( "error" ; 401 )]
End If
```

Try-Catch handling

Exceptions are a popular error handling mechanism in programming languages that support them. FileMaker does not natively support exceptions, but we can organize scripts into analogous patterns with guard clauses that set an error parameter rather than aborting immediately. Once an error is detected at some point in the script, all normal functionality is bypassed by the guard clauses, and errors are address in one section towards the end of the script (like a catch statement).

```
# Script: Find Record ( id ; { layoutName } )Set Variable [$ignoreMe; Value:#AssignScriptParameters]
#
# Check that required parameters are populated
Set Variable [$error; Value:If ( IsEmpty ( $id ) ; 5 ) //Command is invalid (empty parameter)]
#
If [not $error]
        # Go to specified layout
        Go to Layout [$layoutName]
        Set Variable [$error; Value:Get ( LastError )]
End If
#
If [not $error]
        # Find specified record
        Enter Find Mode []
        Set Field By Name [GetFieldName ( GetField ( "id" ) ); $id]
        Perform Find []
        Set Variable [$error; Value:Get ( LastError )]
End If
#
# Handle errors
If [$error //Any error]
        Go to Layout [original layout]
End If
If [$error = 105 //Layout is missing]
        Show Custom Dialog ["The layout " & Quote ( $layoutName ) & " is missing."]
Else If [$error = 401 //No records match the find request]
        Show Custom Dialog ["The record " & Quote ( $id ) & " could not be found."]
End If
Exit Script [Result: # ( "error" ; $error )]
```

Reporting errors between scripts

Both examples above report errors to any calling script via a name-value pair in the script result. This is only one of many possible solutions. Please comment with any modifications or additions you would like to see, or make the changes yourself.

Name-value pair script results:

```
# Sub-script
...
Exit Script [Result: # ( "error" ; $error ) & # ( "otherResult" ; $otherResult)]
```

```
# Calling script
Perform Script ["Sub-script"]
Set Variable [$error; Value:#GetScriptResult ( "error" )]
```

Return-delimited List script results:

By convention, the first item in an ordered list of results should be reserved for any error conditions, since an error may mean that there are no other results to return, and placing the error first means that other results don't have to selectively avoid any higher list positions.

```
# Sub-script
...
Exit Script [Result: List ( $error ; $otherResult )]
```

```
# Calling script
Perform Script ["Sub-script"]
Set Variable [$error; Value:GetValue ( Get ( ScriptResult ) ; 1 )]
```

Global $$ERROR variable:

```
#Sub-script
...
Set Variable [$$ERROR; Value:$error]
Exit Script [Result: $otherResult]
```

```
# Calling script
Perform Script ["Sub-script"]
Set Variable [$error; Value:$$ERROR]
```

Custom functions:

A custom function-based solution for passing error information between scripts would probably also be based on global variables, except that the variables would be managed for developers by the functions.

```
#Sub-script
...
Set Variable [$ignoreMe; Value:SetError ( $error )]
Exit Script [Result: $otherResult]
```

```
# Calling script
Perform Script ["Sub-script"]
Set Variable [$error; Value:GetError]
```