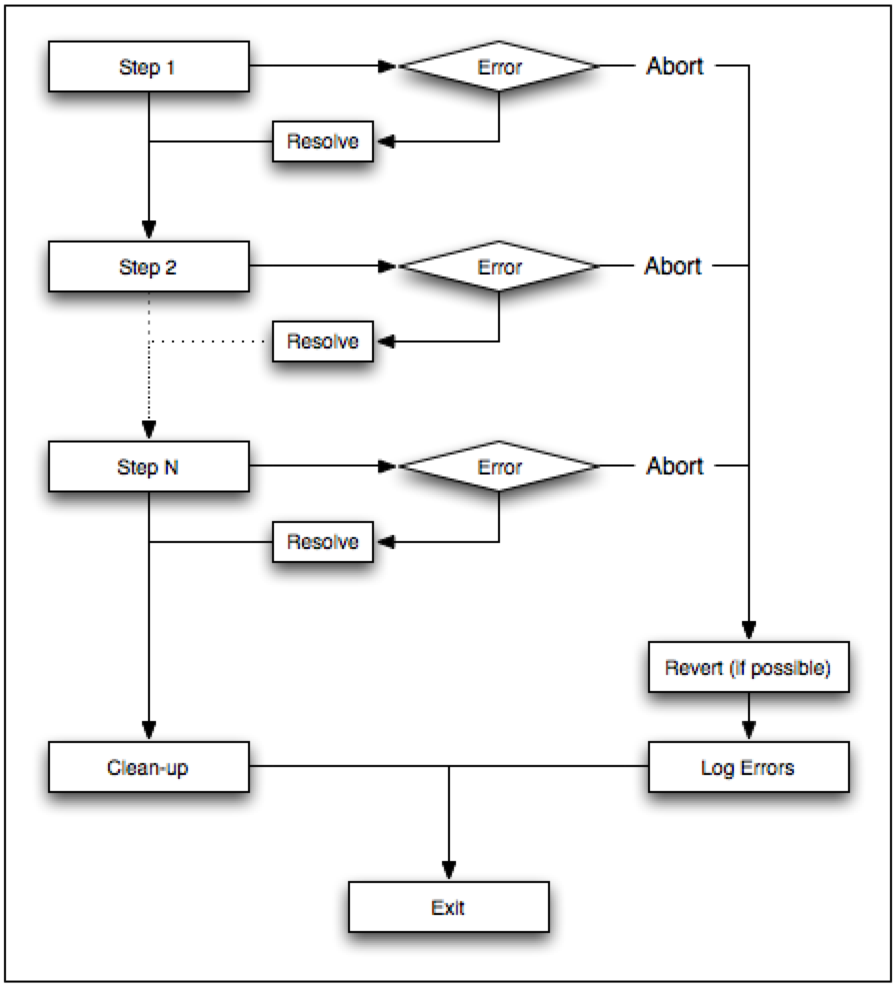
Error Handling

General overview about Error Handling in FileMaker

Error handling is a big deal; and this website could stand to have at least one, if not several, pages dealing with it. The following collected ideas relate to error behavior in scripting, but it may become appropriate to split recommendations into separate pages for Standards, Best Practices, and Accepted Techniques based on the discussion of these issues. Ideas may be added or removed in the process. Please make suggestions.

"Error handling" is an umbrella term for how an application programmatically responds to errors. It's also too vague for a comprehensive discussion of error handling behaviors. A script can react to an error in any of several ways, and may respond to any one error with more than one of these behaviors:

- **Detect the error**, which can include identifying what error occurred.
- **Log the error** — Record that an error occurred, what error occurred, and the conditions of the error (timestamp, script, layout, table, etc.). Error logging is useful for debugging, but does not necessarily play any role in regulating normal application workflow.
- **Report the error** — Notify the calling script or the end user that an error has occurred, and what error occurred. This is not the same action as logging an error, in that reporting an error normally does affect application workflow.
- **Resolve the error** by performing some alternate steps that still satisfy the external expectations of normal execution of that script.
- **Abort on error** — After detecting an error, cease executing the normally expected behavior of the script. A script can perform any reasonable clean-up before aborting and returning control to the calling script or the end user; for example, a script might close any opened off-screen utility windows before exiting.



It is also useful to distinguish between an *error* and a *fault* in execution of a script. A *fault* is something that went wrong; an *error* is what a script detects when a fault happens. For example:

- If your thumb drive explodes while trying to open a file, you might get FileMaker's error 100 (File is missing). The exploding thumb drive is the *fault*; the missing file is the *error*.
- If you loop through a set of records with the Go to Record/Request/Page [Next; Exit after last] script step, you'll get an error 101 (Record is missing) after the step tries to go to the record after the last record, which of course doesn't exist. "Record is missing" is the *error.* There is no *fault;* the loop was supposed to do that.

Some possible guidelines to consider when scripting error behavior:

Resolve an error, or report it, but not both. A script that encountered and resolved an error did manage to complete its expected task, so reporting the error is likely to lead to problems when a user or calling script takes action assuming the task was not completed. So either resolve, or abort and report. This corresponds to the strong exception guarantee.

Use an error dictionary custom function. Some functions already exist that retrieve human-readable descriptions of FileMaker-generated error codes. (For example, ErrorString.) If you use one of these functions, remember to augment it with any additional custom error codes you use in your solutions.

When reasonable, **respect the error capture state that was active when a script was initially called**, i.e., don't present errors to users if a parent script is capturing errors, and therefore assuming responsibility for reporting errors to users. A parent script has broader context, and is better positioned to provide a helpful response to an unresolved error. This idea might be extended by differentiating script roles along similar lines to the model-view-controller model.

```
# Beginning of script
Set Variable [$errorCaptureOn; Value:Get ( ErrorCaptureState )]
Set Error Capture [On]
...
If [$error and not $errorCaptureOn]
        # Report error to user
        Show Custom Dialog [Get ( ScriptName ); "Encountered error: " & $error]
End If
...
# End of script
If [not $errorCaptureOn]
        Set Error Capture [Off]
End If
```

When reporting an error to a user, include steps to take to address the problem. For example, a "Free Lunch" script may ask the user to pre-pay their FileMaker developer $1 million before delivery of free lunches can begin.