Development Guidelines For FileMaker Pro

CLICKWORKS

Version 7.9 February 2023

Cogels-Osylei 36/3, 2600 Berchem
www.clickworks.eu - info@clickworks.eu

Version History

Version 7.9, February 2023: general cleanup of unused rules. Add-on guidelines
Version 7.8, December 2020: new project management chapter, security update.
Version 7.7, December 2019: misc changes, script names, pre-flight checklist.
Version 7.6, May 2019: no more data – interface separation, no more font instructions, perf. script by name
Version 7.4, March 2017: add style name guidelines
Version 7.3, May 2016: add new table, application name.
Version 7.2, December 2015: new way to handle modal dialog windows, new way to deal with Booleans Fields.
Version 7.1, May 2015: scripts must always have a result.
Version 7.0, November 2013: split checklist and guidelines, major review of script naming
Version 6.1, February 2013: added value lists topic.
Version 6.0, August 2012: guidelines Filemaker GO, indexing: define indexing for each field
Version 5.5, May 2012: script comment line for context.
Version 5.4, February 2012: custom functions for global constants + GEN, UTI, GLO, … explained
Version 5.3, September 2011: Layouts: always add developer layout
Version 5.2, May 2011: altered TO color coding advice, minor changes in naming conventions.
Version 5.1, June 2010: no more HUB table, minor changes in naming conventions.
Version 5.0, March 2010: altered naming conventions, added FM11 features. Altered "Error Capture" advice.
Version 4.0, September 2009: altered naming conventions, added FM10 features
Version 3.0, September 2007: simplified naming conventions, added FM9 features
Version 2.5, March 2007
Version 2.0, October 2006
Version 1.0, May 2004

Index

Project Management

- Under-promise, over-deliver.
- When working at the client's office stop earlier to:
  - Check and debug your work
  - Demo your progress to client or power users
  - Prepare a progress report and a planning and discuss with client what to do next and when to do it
- Avoid 'gold-plating': building features that were not explicitly requested by the client.
- Apart from development: these are essential tasks for a developer:
  - Commenting code: think about your co-workers and your future self
  - Organizing project documents: media files, manuals, …
  - Keeping your tickets up-to date
  - Filling in your timesheets.
- Mail or phone?
  - When in doubt, choose phone. Confirm oral decisions via mail.
  - When writing a mail takes more than 5 minutes, choose phone.
- Show, don't tell: plan online meetings with screen sharing, use screenshots, drawings or short video recordings to explain something as visually as possible.

General

- General Naming Conventions:
  - Avoid spaces, use capitalization instead (_CMPid)
  - Keep names short: *CustPhone*, *SupZip* instead of *Customer Phone* and *Supplier Zip Code*

    - Use "_dnr" suffix (Do Not Rename) when object names are hard-coded (for use in evaluate functions for example). Try to avoid hard-coding names as much as possible.
    - The default language for fields, scripts, layouts is English unless agreed otherwise with the customer.

- Avoid hard-coded items. Use global constants instead. Use uppercase for constants. Define global constants as Custom Functions with 'ZK' prefix. For example: 'ZK.BTW' = 21%. Advantages of custom functions:
    - No more typos: always available in functions list
    - Guaranteed read-only: not modifiable at runtime
    - Easily accessible

- Initialize global variables ($$) in InitVariables script if possible. Do not use global variables if other solutions are possible (script parameters, local variables, global constants).

- As a general rule: limit scope as much as possible:
    - No $ signs in Let statements
        - unless you have a damn good reason to do so.
        - Precede that statement with a set var script step to initialize
    - $ vars instead of $$
    - No references to variables in field definitions.

No longer necessary since data migration tool:

- ~~Split your application and create a separate data and interface file. Advantages:~~
    - ~~It's easier to test and develop on a separate data file instead of working on the live files~~
    - ~~Incremental interface versions via releases~~
    - ~~It is faster.~~

Application

- Give your application a name and do NOT use one of the following:
    - Your company name
    - The customer's company name
    - 'FileMaker'
- You want to find a catchy name that users will really use when referring to the app. Your application is not FileMaker nor is it you. Your application will age and look old after a few years. You don't want to look old …

Relationship Graph

- Build an Entity Relationships diagram in a separate file with a file reference to the development file or in a tool like OmniGraffle, Visio, ….

- Use 3 character codes for tables (CMP, CON, …).
- New module (usually when starting new user layout)? Start new Table Occurrence Group (TOG) ~~on a new page in the graph~~
- Recommendation: Put TO groups in alphabetical order by Anchor TO name + do the same with the buoys of each anchor.
- Recommendation: leave enough white space behind a TO. Shift other TOs down if needed. Don't overlap the space behind other TOs. It is much harder to select and fix this afterward than to do that up front.
- Use underscores for each level (CMP_CON_EMP)
- Use Colors only for special TO's (if multiple options apply,use first from list):
  - Red = This TO has option 'Delete related records…' enabled.
  - Other Color: for internal (Interface file) or external TO's (Data File) or to separate TO's from different sources.
  - Grey = default color
  - Green: allow creation of related records
  - Orange: creation + delete
- *Try to avoid sorting in relationships. Mention @ + {asc|desc} + fieldname in relationship name if sorting is applied. Example: CMP_CON@descNameLast*
- For special relationship attributes (filter, range, …), use a tilde (~). Example: CON_OND~ane (Consults – Onderzoeken, type anesthesie).
- Use comments to separate each TOG. Place heading title in comment block.
- November 2013: We discussed anchor-buoy layout options: squid versus spider. Conclusion: we stick with the squid but provide larger horizontal space between the levels to make the relationship join lines easily accessible.
- Use TO name suffixes for tables from External Data Sources:
  - CMPs if Companies source table is an external SQL-table

Create a new table

- Define the table
- Table Name: 3 char prefix + underscore + full name in Title Case
- Rename the auto-created TO to a 3-char anchor code (DO NOT USE in a relationship. The reason is that this TO will always be the first one that is selected in a new calculation).
- Create at least two layouts

- One blank layout for scripting purposes. Do not use any interface elements, script triggers or whatsoever on this layout. A blank layout loads quicker than a layout with data.

- One data layout for development/debugging purposes. Do not use this layout in the user interface.

Fields

- Primary key checks:
  - Name ="__id" to make it appear on top of TO's and as short as possible. This way, the primary key of every table has the same name.
  - Numeric Only
  - Auto-enter serial on creation, unless sync scenarios are in the picture.
  - Minimum validation: unique, not empty
  - Indexing on
  - Should be meaningless to users. Separate application and business logic. Never trust a unique key that has meaning to users. Sooner or later it will change.

- Foreign key checks
  - Name starts with underscore "_" (_CMPid) to make it appear just below primary keys in TO's.
  - Name = prefix from related table + "id"
  - If required: validation not empty
  - Indexing on

- *Multikeys*
  - Add 's' to key name, example: _CMPids

- Required fields in each table (tip: copy/paste the template table)
  - Primary key
  - zzzCreatedOn (host timestamp!)
  - zzzModifiedOn (host timestamp!)
  - zzzCreatedBy
  - zzzModifiedBy
  - zzk0
  - zzk1
  - zzsListOfIDs
  - Use Default Fields in FileMaker 17 or higher.

- Naming conventions: see ClickWorks Naming Conventions.

- Add comments to uncommon fields. Consider using Field Comments for tooltips.

- Validation: consider validation options: not empty, in range, data type, etc…

- Indexing: define indexing for each field (do not leave this decision to the user, because when indexes are created at run-time, this will be a tedious experience for users).

  - One Quicksearch will index ALL fields in a table if indexing has not been turned off!
  - Turn indexes ON when recovering a file

- Calculations: consider making them stored or unstored. For example: calculations based on today's date do NOT refresh if they're stored.

- Boolean fields:
  - define as number
  - auto-enter GetAsBoolean(Self)
  - validation: always validate 1 or 0

Layouts

- Standard Folders, hidden from user:
  - Developer: table view layouts to show data (CMP_Company, CON_Contact)
  - Internal: empty layouts for use in scripts (CMP, CON…)
  - Under Construction: all layouts you don't want to show to users yet
  - Cleanup: layouts to check for removal (append '__DELETE' to the layout name as well and check them with inspector for references before really deleting them)
  - Be aware that FileMaker only can display a limited number of folders.

- Create a template layout with a grid, styles, … Check the smallest screen size.

- When adding objects to a layout, always align them (tip: use Guides). Use shortcut keys:
  - Alt
  - Shift
  - CTRL/Command

- If hidden from the user navigation, use TO-prefix: CMP_popup, CON_picker.

- Always create layouts used only for scripting (only TO name: CMP, CON,…). Never use interface layouts in scripts unless explicitly stated in script comments (to avoid surprises with script triggers, commit etc…)

- Use tooltips to help users with interface elements. Tip: you can automate this and use (part of the) field comments in your tooltips. This allows documenting the solution without modifying every single tooltip. When copying a field object, check the tooltip.

- Field colors: make a visual distinction between:
  - Browse
  - Find access
  - No access

- Font: Use a cross-platform, compact font which displays numbers well (distinction between 0, 6 and 8): Tahoma, Calibri, Verdana …

- Fields:
  - Padding: left and right in input layouts
  - This allows to group fields together without leaving spaces between the field borders.

- Field Labels, be consistent on:
  - Using a * for mandatory fields
  - Align right (next to field = visual highway)
  - Title Case
  - Colon (example: Company Name:)

- Column labels in List View or Portals: use the same width as the corresponding field object. This makes resizing and aligning easier.

- Tab Order:
  - Consider including/excluding buttons from tab order
  - Adjust the options to go to the next Field: turn Tab, Return on and turn Enter off.

- Only turn on Spell Check when needed.

- Only turn on Quick Search when needed.

- Numbers:
  - Align right in lists.
  - Currencies: include the currency symbol.
  - Use system formats for decimal and thousand separators.

- Report tips:
  - Make Titles and summary titles big enough
  - Multiple subsummaries: use indenting or find another way to make a difference between the different titles
  - Remove column titles if column contents are obvious
  - Consider the sort order: try to create groups of data
  - Overlap fields with 1 pt when sliding fields
- Check Layout resizing and object anchors (trick: zoom out to 75% and check object alignments)

Themes and Styles

- Themes:
  - One theme per device type/purpose: Touch, PC/Mac, Reporting.

- Do as much as possible in the general purpose theme (backgrounds, colors, ...) before duplicating the theme for use on other devices.
- Create a template layout with most of the objects used in the app.

- Styles:
  - Use defaults for most commonly used (form view edit).
  - Name other styles to describe their function and why they differ from the default. Example: EDB.LST.ReadOnly (Editbox, List View, Read Only), EDB.LST.ReadOnly.Inactive. This way, incremental additions are visible as such.
  - Check all the states when creating a new style.
  - Create a new style for every modification to one of the existing styles. Don't go for 'exceptions'.

Scripts

- Never assume:
  - **Check** context, variables etc… if required
    OR
  - **Establish** context

- Use the template script or duplicate existing script.

- Script names should have the following structure: Entity.ScriptName (parameterList) = resultsList
- TODO: rephrase: If a script does not go to the context of that entity, use Entity:ScriptName (colon).
  - Example: DOC:Mail, assumes logic works on the DOC Table Occurrence.
    Attention: when using 'Perform Script By Name', Use the 'script.executableName' custom function to retrieve the script name by its Id. Reason: a colon in a script name does not work unless you use "::" as prefix. Example: Perform Script By Name ("::DOC:Mail").

- Entity = A Table Occurrence or logical Category. When the functionality is directly associated with a table occurrence, use this as Entity, otherwise use a category the functionality applies to. [TODO: JAN ZELENKA -> REVIEW]
  - Examples: CMP.New, CON.Edit(CONid)
  - Category examples:
    - CMP, CON, PRO: module-specific scripts (companies, contacts, projects, …)
    - GEN: *General*. These scripts can be copy/pasted in other apps. They require none but framework-context (like a Windows table for the windowhandler). These script are meant to be called as subscripts from your main scripts to perform common technical tasks.
    - Sec: *Security*. Scripts to handle security account creation/deletion. Can also set security parameters.
    - Update: Wnd., Rec., Nav, …

- ScriptName = A readable name for the functionality the script provides.

- Script suffixes (optional). ScriptName
  - + _NP for scripts that do NOT initialize a Rocket Process script.
  - + _I : has user interaction
  - + _L : changes the active layout
  - + _C: performs a commit

- ParameterList = A list of parameters inside brackets separated by semicolon. Optional parameters are put to square brackets. If a parameter accepts only values form a predefined set, put an equal sign behind it and list the values in curly brackets separated by a pipe. If there is not enough space to list all the parameters, use 3 dots to indicate there are more and document in the script. Examples:
  - Parameters, some optional: (intINVid ; strDescription [;fltDiscountRate])
  - 2 optional parameters where second one makes sense only if the first one is specified but is still optional: ([intINVid [;intINLid]])
  - An optional parameter with a predefined set of values: ([intDocType = { 1|2|3 }])

- ResultsList: the result list is subject to the same rules as the parameter list with one exception:

  - When a script doesn't return a structured value but directly a value, the possibilities should be listed. Example: Entity.Scriptname = { True|False }

- Script Comments:
  - First lines = comment lines with date, author and general purpose of script
  - Add a line with the Context (TO name) that is required for the script or mention 'Any Context' if applicable. If helpful add comment like 'User Clicks button X in portal Y'.
  - Add comment lines for each modification with date + author
  - Explain loops, complex IF statements, complex set field/variable statements, etc…
  - Explain what you WANT to do not what you ARE doing.
  - Explain incoming/outgoing parameters
  - Optional: add a sample call to the script as a template. This is a disabled Perform Script example with all parameters.

- Use "Set Error Capture" only when strictly needed! Ex.: Perform Find where an empty found set can occur. We want to know as early as possible if a script fails.

- Set "Allow User Abort" to Off at the start of the script unless you really want the user to interrupt your script. In that case, consider the 'garbage collection' technique with an OnTimer script to guarantee a smooth landing. [TODO: JORIS, add a footnote with link to blog article]

- Never delete script steps from another author without checking first. Instead, disable them and optionally add start and end comment line.

- Script structure:
  - Init phase: set variables on the top of a script makes it easier to read.
  - Piecemeal variables: prepare complex expressions in different variables
  - Validation phase. Handle it if something's wrong. Examples:

        - Do we have all required script parameters?
        - Are we in the right Mode?
        - Are we on the right layout?
        - Has current record been committed or is it still in edit mode? (Rec.Commit)
        - Has this user the correct access rights to do this?
        - Etc…

- Common mistakes:
    - Do not forget to use Commit Record after creating/updating records.
    - Go To Related Record doesn't work if no related records exist. Script stays in the same layout as before. Always test if related records exist beforehand: error if no related records exist (Get (Lasterror) <> 0). Caution: with 'match found set' you get the following error codes:
        - 0: found matching records and the active record in source matched a record in destination
        - 101: found matching records but the active record in source did not match a record in destination
        - 401: there were no related records in destination table
- Disable the checkbox to display the script, unless you have a reason to show it in the Scripts menu

- Avoid long IF… ELSE … END IF constructions (ex. For error checking). This allows you to separate the 'real' script from the conditions/error checking that's usually done at the start of the script.

- Messages:
    - Should be clear and polite.
    - Errors: distinguish minor/critical warnings.
    - Explain what happened and what the user should do next.
    - If user should contact helpdesk, display at least script name and error code. Better yet, divert to a dedicated error handling script that gathers as much context-information as possible.

- Avoid linear scripting that traps the user in a series of actions. Always provide an escape route or allow users to cancel an operation.

- Popup dialog layout with multiple buttons :
    - Call the popup window and set Window Setting to 'modal'.
    - All dialog buttons call the Wnd.ExitModalScript with script parameter reflecting the button action. The button's setting must be 'Resume Script'.
    - The Wnd.ExitModalScript passes the button parameter back to the calling script without the need for an extra looping script.
    - Centralize the button handling in the main script.
    - If necessary: trap recordCommit via Script Trigger.

- A script must always return a result. The reason is that Get (ScriptResult) returns the result from the last script that has a result, not always the last one. Make sure every script ends with 'Exit Script' and with a result. This may even be an empty result (empty quotes).

Valuelists

Make a distinction between:

1. Hard-coded general lists, to be used in different places. *Ex. _1, _OK*
2. Hard coded items used in specific context: ex. Customer Status: *CUS.Status*
3. Related values: example, TO-chain.FieldName: Customer->Invoice Date: CUS_INV.InvoiceDate

Add-Ons

Numerous errors can occur when creating and also when integrating add-ons in your solution and not all errors are handled gracefully by FileMaker. Examples: what happens when the add-ons contains scripts, custom functions etc… with the same names or object Ids as existing objects in your solution?

Therefore we recommend the following security measures before adding an add-on to your solution:

- Make a backup first
- Make sure no other users are connected. We had an issue with an add-on only partially integrated because of a locking conflict with another developer.
- Check if no functions or scripts have been overwritten by objects from the add-on.

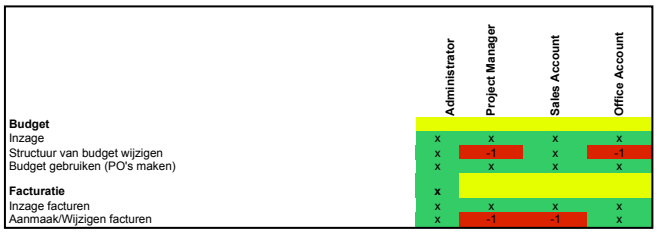
Testing and debugging

- Never deliver a script without first running it yourself.
- Keep a notepad at hand and jot down immediately every non-conformity you notice. Solve the problems right away or log them as to-do's in a ticketing system.
- Know the weak spots :
    - Context issues : current layout, mode, privilege set, status of globals and variables
    - Data entry by user : double check
- Test on the platforms (os/device type) that the app will be used
- Test every single combination of buttons that the user encounters on his path through a procedure. If required, make a matrix with all possible combinations and test each combination one by one.
- Test with all security profiles.
- Record a short video demo to explain a new feature.
- How to test user data entry :
    - Test with invalid entries : text, numbers, symbols
    - Test with extremes : nothing, very small numbers, very large numbers

Security

Make a security matrix :

- Rows : context and action (“edit company records”)
  - Complex: Read, Edit, Create, Delete
- Columns: privilege set
- Cells: Simple: 0 (not allowed), 1 allowed, empty (to discuss)
- Review this matrix with the client



	Administrator	Project Manager	Sales Account	Office Account
Budget				
Inzage	x	x	x	x
Structuur van budget wijzigen	x	-1	x	-1
Budget gebruiken (PO's maken)	x	x	x	x
Facturatie	x			
Inzage facturen	x	x	x	x
Aanmaak/Wijzigen facturen	x	-1	-1	x

- Create at least the following accounts:
  - Management Backup Account (Madmin): ask management
  - Project Admin-Account (Padmin)
  - Individual Developer Accounts
- Consider creating separate accounts for:
  - One backup account for the client power user if they create accounts via scripts.
  - Server account for server-side scripts.
    - DO NOT use Madmin or Padmin.
    - Avoid Full Access.
  - Service accounts for APIs, Migration, ODBC, etc…
- If client has Full Access account, create a non-administrator account too for daily use. Avoid having users working all day with admin access.

FileMaker GO

General

- Consider modifying the default extended privilege 'fmreauthenticate10'.

Scripting

- Check script steps. Some script steps don't work on FileMaker Go or behave differently. See tech guide.
  - Tip: use the script compatibility button in the Script Editor.
- Consider setting 'User Abort OFF'. Else, if a user taps the screen: user gets abort dialog.

- Use 'Go To Layout' instead of 'New Window' because this is always visible and causes flickering on iOS.

Layout

- Use Empty layouts in scripts: because layouts with fields become visible on iOS when running scripts.
- Before beginning developing the layouts for iPad. check how they want to use it: landscape, portrait or both. Makes a huge difference for the layout designs.
- Height of lists: minimum 45 pt for touch, otherwise it becomes difficult to select right record.
- Tip: always select 'show vertical scrollbar' for input text fields. In that way the field does not get larger when you enter the field.

Pre-flight checklist

General

1. Missing items and broken references check (FMPerception)

Scripts

1. Check script visibility. It's ugly when a user sees developer scripts in the Scripts Menu.

Layouts

Check layout visibility: make sure the user does not see layouts meant for development purposes.
Apply the following checks on each layout:

1. Object anchoring
2. Tab Order
3. Include for Quick Search
4. Spell-Checking
5. Styles: Theme Changed?
6. Placeholder Texts
7. Tooltips