Development Guidelines For FileMaker Pro

Version 7.0 November 2013

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

Version History

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

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. New 2012: define global constants as Custom Functions with 'zzk_' prefix. For example: 'zzk_BTW' = 21%. Advantages of custom functions:
  - No more typos: always available in functions list
  - Guaranteed read-only: not modifiable at runtime
  - Easily accessible

- Initialise 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
  - $ vars instead of $$
  - No references to variables in field definitions.

- 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.

- 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

Relationship Graph

- Build an ER 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, …) and keep a dictionary.
- New module (usually when starting new user layout)? Start new Table Occurrence Group (TOG) on a new page in the graph
- 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.

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.
  - 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
  - zzzCreaTS
  - zzzModTS
  - zzzCreaName
  - zzzModName

- 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 indexing is created at run-time, this will be a tedious experience for users).
  - Be careful when turning indexing off permanently.

- Calculations: consider making them stored or unstored.

Layouts

- Standard Folders, hidden from user:
  - Developer: empty layouts for use in scripts
  - 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)

- Duplicate existing template layout

- When adding objects to a layout, always align them (tip: use T-squares)
  - Alt
  - Shift
  - CTRL/Command

- If internal, use TO-prefix: CMP_popup, CON_picker

- Always create developer layouts (only prefix: 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.

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

- Font: Use a cross-platform, compact font which displays numbers well (distinction between 0, 6 and 8): Tahoma 9pt (PC) or 10pt (Mac)

- Fields:
  - Internal spacing: 4pt left and right in input layouts
  - Internal spacing: 2pt in reports
  - This allows to group fields together without leaving spaces between the field borders.

- Field Labels
  - Use * for mandatory fields
  - Title Case
  - Align Right (next to field = visual highway)
  - Add colon (example: Company Name:)

- Check tab order for entry fields
  - Consider including/excluding buttons from tab order

- >FileMaker 9: consider turning spell check on/off for certain fields (ex: no spell check on phone/fax fields)

- >FileMaker 11: consider turning quick find on/off for certain field (watch for unindexed fields!)

- Date format: dd/mm/jjjj, align right

- Numbers:
  - Align right
  - Currencies: € 1.000,00

- Check horizontal/vertical spacing

- 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

- Check Layout resizing and object anchors

Scripts

- Use the template script or duplicate existing script

- Script names should have the following structure: Entity.ScriptName(parameterList)=resultsList

- 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.
    - TO 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.
        - UTI: *Utility*. These scripts can be copy/pasted in other apps. They require no context. These script are meant to be run directly from the scripts menu by a developer (to get window sizes, to unlock the status area, to re-login, etc…)
        - GLO: *Global*. These scripts are application-specific but are meant to perform global tasks, depending on the context if needed. Examples: GLO_NewRecord, GLO_DeleteRecord.
        - INI: *Initialization*. Scripts to run at Startup and/or Shutdown.
        - SEC: *Security*. Scripts to handle security account creation/deletion. Can also set security parameters.

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

- 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 a pipe. If there is not enough place 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 addition. When a script doesn't return a value dictionary string but directly a value, the possibilities should be listed. Example:
    - {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 assumed for the script or mention 'No Context' if applicable. If helpful add comment like 'User Clicks button X in portal Y'.
    - Add comment line 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 if not obvious

- 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 OnTime script to guarantee a smooth landing.

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

- Script structure:
  - Ini phase: set variables
  - Validation phase. Exit script 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? (GEN_TryToCommit)
    - 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 before using: error if no related (getlasterror<>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

- Consider activating the option "Run with Full Access Privileges". Add asterisk (*) to script name when option is enabled.

- 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.

- Use Replace instead of Loop, it's faster but mind record-locking issues in multi-user context.

- Globals are stored locally on the client machine. Mind this when developing client-server applications!

- Error messages:
  - Should be clear and polite.

  - 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.

- Consider blocking access to the 'Window' menu for certain procedures. Use a general looping script.

- Avoid linear scripting that traps the user in a series of actions. Always allow users to cancel an operation. Avoid using the pause/resume option too much.

- Popup dialog layout with multiple buttons :
  - Use one script with a loop to lock the user in the dialog.
  - Set all buttons to 'Exit Script'.
  - Add a different script return parameter to each button.
  - Centralize the button handling in the main script.
  - If necessary: trap recordCommit via Script Trigger.

Valuelists

- *GEN*: all hard-coded general lists, to be used in different places. Ex. GEN_1, GEN_OK

- *TO-name::FieldName*: hard coded items used in specific context: ex. KLA::Status

- TO-chain::FieldName: for related values: KLA_FAC::factuurdatum

- TO-chain with tilde if relationship is special (not traditional pk - fk pattern). Ex. Contacts From Supplier on Order: ORD_CON~supplier::LastName

- Use Underscores in fieldname when combining multiple fields: ex. ORD_CON~supplier::LastName_FirstName

Testing and debugging

- 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 Bug Tracker system.

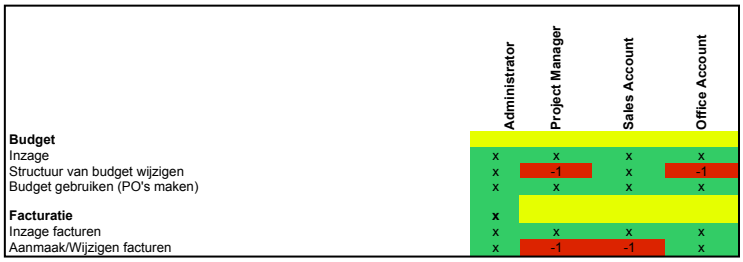
- Know the weak spots :
  - Context issues : current layout, mode, privilege set, status of globals and variables
  - Data entry by user : double check

- How to test user data entry :
  - Test with invalid entries : text, numbers, symbols
  - Test with extremes : nothing, very small numbers, very large numbers

- Test every single combination of buttons that the user encounters on his path trough a procedure. If required, make a matrix with all possible combinations and test each combination one by one.
- Test with all security profiles.

Security

- Make a security matrix :
  - Rows : context and action ("edit company records")
  - Columns: privilige set
  - Cells: 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:
  - ClickWorks-login
  - One backup account for the client power user if they create accounts via scripts.

- If client has administrator account, create a non-administrator account too for daily use. Avoid having users working all day with admin access.

- Challenge: Security and interface-data separation: It's not possible to have interface scripts run with full access on the data file. But sometimes in a specific context, you want ordinary users to perform extraordinary tasks and they temporarily need more privileges than their security profile allows them.

  Workaround: Create a reference from the Data file to the Interface file (you probably already needed one). Tie data table security to a global field in the interface file, example: 'zgOverrideSecurity'. When the switch is on, the user can edit/delete/…

  Set the switch on/off using scripts in the Interface file.

  Caution: don't forget to set the switch to 'OFF' if special record access is no longer needed!

FileMaker GO

General

- Consider adding the extend privilege 'fmrestorelogin'.
- Consider if it is really necessary to have separate interface and data-file for iPad and desktop version. If this is the case: know that you always will have to update both files and you will have to do double work when business logic is added. You will have to make double TO's (in both interface files) and have to make double table's and fields (in both data files). And problems can occur with sync when fields are not added in the right order, because FM always looks at the sort by creation order and matches not automatically on name! Even if you've specified so in the import script step.
- Conclusion: if not necessary i think it is better to have one IF file and have one data file for both version, iPad and desktop. (for SolePad we have this problem now)

Scripting

- Check script steps. Some script steps don't work on filemaker go or behave differently. See tech guide.
- Consider using 'User Abort': when not on: when a user taps the screen when a script is running: user gets abort dialog.

Layout

- DEV layouts: no fields on layout: when you use them in scripts you will see the fields if there where fields on the dev-layouts.
- 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 40px 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 unwanted larger when you enter the field.
- If possible avoid portals: not very handy for scrolling. Only when the portal rows don't have a 'button' script: when you click on the portal: the portal is active and you can scroll in the portal. Otherwise you will have to use a pen to use the vertical scrollbar.