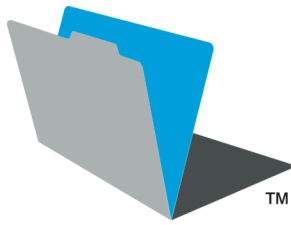


FileMaker Development Conventions



FileMaker®
Development Conventions

©2005 FileMaker, Inc. All rights reserved. FileMaker is a trademark of FileMaker, Inc., registered in the U.S. and other countries. The file folder logo and ScriptMaker are trademarks of FileMaker, Inc. All other trademarks are the property of their respective owners. Product specifications and availability subject to change without notice.

FileMaker documentation is copyrighted. By downloading from the FileMaker website you agree not to make additional copies or distribute this documentation without written permission from FileMaker.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, AND FILEMAKER, INC., DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, OR THE WARRANTY OF NON-INFRINGEMENT. IN NO EVENT SHALL FILEMAKER, INC., OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF BUSINESS PROFITS, PUNITIVE OR SPECIAL DAMAGES, EVEN IF FILEMAKER, INC., OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY. FILEMAKER MAY MAKE CHANGES TO THIS DOCUMENT AT ANY TIME WITHOUT NOTICE. THIS DOCUMENT MAY BE OUT OF DATE AND FILEMAKER MAKES NO COMMITMENT TO UPDATE THIS INFORMATION.

Introduction

Programming in FileMaker® Pro is quite different from programming in other development environments. FileMaker developers have the freedom to rapidly create and modify applications without having to deal with many of the constraints or concerns other developers experience. When employing an interactive approach to design, a tool like FileMaker lets you modify and extend functionality with little regard to dependencies elsewhere in the solutions. However, the more complex the solutions, the more difficult it is to maintain.

For instance, without leaving the development context and navigating numerous dialogs, it's impossible to understand essential information about FileMaker Pro's primary programmable object: the field. What is its data type? What is dependent on it? How does it store data? What is its location or participation in the relationship graph? How will it behave in a script, on a layout, in a calculation, or in a value list?

FileMaker Pro's flexibility offers numerous and unique ways to approach development. This has led to a plethora of redundant but different standards. Many developers use standards; but each uses their own, tied to their specific development methodology and unique preferences. Some developers have publicly published their standards, but adoption has been limited.

No naming convention can address every conceivable development methodology and solution design. Furthermore, it would be counter-productive to try to establish such a rigid rule set. What you need are baseline recommendations to establish a set of criteria you should consider for any naming conventions. This baseline would be a starting point you can extend into a richer convention to meet the needs of more demanding solutions and design methods.

In response to this need, FileMaker Inc. established a committee that brings together experts within the FileMaker development community to create awareness and establish guidance for naming conventions. This dedicated group, known as the FileMaker Development Conventions Advisory Committee (FDCAC), represents a cross-section of expert, FileMaker Solutions Alliance (FSA) partners and associate members from around the world. Collectively, they represent hundreds of years of experience working on individual and enterprise projects and packaged solutions development. This committee provides a wide variety of opinions and practices, representing a broad spectrum of approaches to conventions rather than one specific view.

The objective of the FDCAC was to create the FileMaker Development Conventions (FDC) white paper, the goal of which is to promote consistency and professionalism among FileMaker solutions.

The FileMaker developer base is extremely diverse. The approachability of the product makes it a clear choice for novice application developers, and the rapid application development nature of the tool is attractive to more seasoned developers, too. The amount of effort a developer needs to apply to naming conventions is different at each end of the spectrum.

The FDC is designed to provide all levels with some degree of interest, but is especially designed for the intermediate developer. The FDC isn't targeted at the FileMaker Pro beginner. However it also isn't for extremely advanced solution developers because the convention guidance might not be sufficient. Likewise, commercial or packaged solutions developers might not find the FDC directly applicable because the complexity and architecture might require more stringent and specific conventions. But these developers can leverage some of the wisdom in the document to further refine their specific conventions.

The general FDC recommendations give developers a template to work with. The “standards-aware” recommendations don’t address every conceivable architecture or methodology. However, they do provide well-founded guidance on where to start, which is likely to evolve over time.

The FDCAC approach was to identify and further define the specific problems naming conventions are designed to address. With the problems identified, the committee then looked for commonalities among the plethora of variants. In many cases, simply defining a set of consistent guidelines addresses the concern. However, there are other topic areas where a particular style or development methodology dictates the convention to apply. In these cases, the FDC has elected to provide recommendations for general standards and additional considerations that provide insight on how some developers have approached these more advanced areas.

The FDC is made up of multiple topics that cover almost any aspect within the FileMaker development environment where a developer might apply naming conventions. Topics include:

- File naming
- Table naming
- Field naming
- Table Occurrence naming
- Layout naming
- Calculation documentation and formatting
- Value list naming
- Custom Function documentation and formatting
- Account naming
- FDC adherence documentation

Each section has three sub-sections, the first of is the Problem Definition section. This section identifies the key areas where a developer might apply a convention:

- Address a shortcoming of the development environment,
- Avoid all-too-common-convention pitfalls, or
- Apply consistency, understanding, and documentation.

The second section of the FDC discusses recommendations for being what the FDCAC calls “standards-aware.” This provides a template, a shell if you will, that developers can use when applying conventions to their projects.

Last, the document discusses a few examples of how FDCAC members have approached conventions in each topic area. The purpose is to provide a sample of more advanced concepts and extensions above and beyond what the FDC “standards-aware” provides.

Although FileMaker believes all levels of developers can benefit from the information within the FDC, the greatest benefactor is the FileMaker community. This is the first step in an evolving process.

Clearly, this document isn’t for everyone. If you’re just starting with FileMaker, we encourage you to keep doing what you’re doing, but we also encourage you to peruse the FDC to become familiar with its concepts. If you’re building the next killer application and you’ve developed your

own conventions to address more advanced methodologies, this is somewhat of a review. However, we hope you consider the information and try to utilize it whenever you can.

For all those developers somewhere in the middle, you have the knowledge of expert FileMaker partners and associates available to you, so use them. The FDC document's guidance can help you accelerate the deployment of professional and durable solutions.

Preface

The FileMaker Development Conventions document (FDC) is in no way intended to be considered the “right” or mandated way you should develop your FileMaker solutions. It would be counterproductive to try to establish such a rigid rule set. After all, one of the hallmarks of FileMaker is its flexibility. Neither FileMaker nor anyone else can mandate the methodology of your solution or your naming convention. We also do not believe that any one or any single organization has sufficient perspective required to establish a convention either. The effort put forth here is about establishing a first step in what is sure to be an evolving process. Its success is not dependent upon any one individual. The FileMaker community, as a whole, needs to gravitate towards this effort and collectively nurture it throughout its evolution. It is clear there is motivation within the FileMaker development community to utilize a standards framework, as we all know the benefits for communication and durability.

There are considerable advantages to the effort. Collaborative development teams can work with one another easier. Developers from different organizations may be able to quickly understand the others code, and even their own code years later. It enables a more consistent and understood approach to creating templates and open solutions that are shared. It can establish clear and consistent documentation. And some of the simplest guidelines can prevent common solution evolution problems as well as compatibility issues.

FDC version 1.0 was developed with the FileMaker 7 product line, as the basis of its content. Some consideration was given to FileMaker 8 where easily applicable, however, the FDC specifically chose to not address, in too much detail, the new features available in the FileMaker 8 product line. This decision was in line with one of the primary goals, which was to provide “real-world” based guidance; this did not exist before the writing of this document. Future FDC documents will provide additional guidance should it be necessary.

New and seasoned developers alike will benefit from the birth of the FDC. As the FileMaker product evolves, the myriad ways we approach problems will likely change as well. The community needs this continuing effort and with the support of the FileMaker community, like the FileMaker product line, it is likely to become better with each version.

Acknowledgements

We want to whole-heartedly thank each of the following companies for their liberal consent of time and profound knowledge donated to this project.

| | |
|--------------------------------|--|
| Beezwax | http://www.beezwax.net Vince Menanno Rick Aguire |
| Core Solutions | http://www.coresolutions.ca Steve Hearn |
| Dataworks | http://www.dataworks.ca James Hea |
| DataWaves International | http://www.data-waves.com Peter Makin Colleen Hammersley |
| FileMaker | http://www.filemaker.com |
| Geist Interactive | http://www.geistinteractive.com Todd Geist |
| InResonance | http://www.inresonance.com Corn Walker |
| iSolutions | http://www.isolutions.com Cris Ippolite |
| Management Counseling Services | http://www.fmp-power.com Steven Blackwell |
| The Moyer Group | http://www.moyergroup.com Chris Moyer |
| New Millennium Communications | http://www.nmci.com Danny Mack |
| Soliant Consulting | www.soliantconsulting.com Steve Lane Scott Love Rodger Jacques |
| The Support Group | http://www.supportgroup.com Chad Novotny |

FileMaker Development Conventions

Table of Contents

| | | |
|----------|--|-----------|
| 1 | FILES | 9 |
| 1.1 | OBJECTIVES | 9 |
| 1.2 | PROBLEM DEFINITION | 9 |
| 1.3 | STANDARDS AWARE GUIDELINES (FILES) | 11 |
| 1.4 | ANCILLARY CONSIDERATIONS: | 13 |
| 2 | TABLE NAMING | 14 |
| 2.1 | OBJECTIVES | 14 |
| 2.2 | PROBLEM DEFINITION | 14 |
| 2.3 | STANDARDS AWARE GUIDELINES (TABLES) | 16 |
| 2.4 | ANCILLARY CONSIDERATIONS | 16 |
| 3 | FIELD NAMING | 18 |
| 3.1 | OBJECTIVES | 18 |
| 3.2 | PROBLEM DEFINITION | 19 |
| 3.3 | STANDARDS AWARE GUIDELINES (FIELDS) | 20 |
| 3.3.1 | General Fields | 20 |
| 3.3.2 | Key/Match Fields | 21 |
| 3.3.3 | Utility Fields | 23 |
| 3.3.4 | Field Notation Reference | 24 |
| 3.4 | ANCILLARY CONSIDERATIONS | 25 |
| 4 | TABLE OCCURRENCES | 26 |
| 4.1 | OBJECTIVES | 26 |
| 4.2 | PROBLEM DESCRIPTION | 27 |
| 4.3 | STANDARDS AWARE GENERAL GUIDELINES (TABLE OCCURRENCES) | 28 |
| 4.3.1 | Functional Spider Grouping (FSG) Method | 29 |
| 4.3.1.1 | Functional Spider Grouping Pros | 30 |
| 4.3.1.2 | Functional Spider Grouping Cons | 31 |
| 4.3.1.3 | Functional Spider Grouping: Standards Aware Guidelines | 31 |
| 4.3.2 | Functional Table Occurrence Groups (FTOG) Method | 32 |
| 4.3.2.1 | Functional Table Occurrence Grouping (FTOG) Pros | 32 |
| 4.3.2.2 | Functional Table Occurrence Grouping (FTOG) Cons | 33 |
| 4.3.2.3 | Functional Table Occurrence Groups: Standards Aware Guidelines | 33 |
| 4.3.3 | Anchor Buoy / Hierarchical Table Occurrence Grouping (HTOG) Method | 37 |
| 4.3.3.1 | Anchor Buoy / (HTOG) Pros | 37 |
| 4.3.3.2 | Anchor Buoy / (HTOG) Cons | 38 |
| 4.3.3.3 | Anchor Buoy / HTOG Standards Aware Guidelines | 38 |
| 4.4 | ANCILLARY CONSIDERATIONS | 40 |
| 5 | LAYOUTS | 42 |
| 5.1 | OBJECTIVES | 42 |
| 5.2 | PROBLEM DEFINITION | 43 |
| 5.3 | STANDARDS AWARE GUIDELINES (LAYOUTS) | 44 |
| 6 | CUSTOM FUNCTIONS | 46 |
| 6.1 | OBJECTIVES | 46 |
| 6.2 | PROBLEM DEFINITION | 47 |
| 6.3 | STANDARDS AWARE GUIDELINES (CUSTOM FUNCTIONS) | 48 |
| 6.3.1 | Public Custom Functions | 48 |
| 6.3.2 | Private Custom Functions | 48 |
| 6.3.3 | Custom Function Parameters | 49 |
| 6.3.4 | Custom Function Naming Examples | 50 |

| | | |
|-----------|--|-----------|
| 6.3.5 | Custom Functions Documentation | 50 |
| 6.3.6 | Custom Function Formatting | 51 |
| 6.4 | ANCILLARY CONSIDERATIONS | 51 |
| 7 | SCRIPTS | 52 |
| 7.1 | OBJECTIVES | 52 |
| 7.2 | PROBLEM DEFINITION | 52 |
| 7.3 | STANDARDS AWARE GUIDELINES (SCRIPTS) | 53 |
| 7.3.1 | Script Names | 53 |
| 7.3.2 | Variables | 53 |
| 7.3.3 | Script Documentation | 53 |
| 8 | CALCULATIONS | 55 |
| 8.1 | OBJECTIVES | 55 |
| 8.2 | PROBLEM DEFINITION | 55 |
| 8.3 | STANDARDS AWARE GUIDELINES (CALCULATIONS) | 56 |
| 8.3.1 | Variables | 56 |
| 8.3.2 | Calculation Block Header Commenting | 56 |
| 8.3.3 | In-Line Calculation Commenting | 57 |
| 8.3.4 | Calculation Formatting | 57 |
| 8.3.4.1 | Formatting Example #1 | 57 |
| 8.3.4.2 | Formatting Example #2 | 59 |
| 8.4 | ANCILLARY CONSIDERATIONS | 62 |
| 9 | VALUE LISTS | 63 |
| 9.1 | OBJECTIVES | 63 |
| 9.2 | PROBLEM DEFINITION | 63 |
| 9.3 | STANDARDS AWARE GUIDELINES (VALUE LISTS) | 65 |
| 10 | ACCOUNTS & SECURITY | 66 |
| 10.1 | OBJECTIVES | 67 |
| 10.2 | PROBLEM DEFINITION | 67 |
| 10.3 | STANDARDS AWARE GUIDELINES (SECURITY) | 69 |
| 10.3.1 | Privilege Sets | 69 |
| 10.3.2 | Internally Authenticated Account Names | 69 |
| 10.3.3 | Externally Authenticated Account Names (Group Names) | 69 |
| 10.4 | ANCILLARY CONSIDERATIONS: | 71 |
| 11 | ADHERENCE | 73 |
| 11.1 | STANDARDS AWARE CONVENTION GUIDELINES | 74 |
| 12 | APPENDIX A - SQL RESERVED WORDS | 75 |
| 13 | APPENDIX B - CHARACTER USAGE CHART | 83 |
| 14 | APPENDIX C – TERMINOLOGY & DEFINITIONS | 85 |
| 15 | APPENDIX D - SYNTAX LEGEND | 87 |

I Files

The design needs and architecture of a solution will have impact on whether it's made up of a single file or multiple files. Designs may include a single file, which contains all the tables, or separation architectures, which can split data and interface into different files. Solutions may contain multiple utility files used for reporting or solution modules. In any case, a solution is either a single file or multiple files. For each of these files you should consider a number of factors during the naming process.

A character set should be used that will not encumber connectivity to the solution from technologies such as ODBC, JDBC, and XML. Select a case and be consistent with it. In conjunction with case, consider how words within the name will be separated. In addition, consideration must be made to how the file name itself will be syntactically separated from any meta-data, prefix or suffix, which is referred to as syntax separation throughout this document. Selecting if one will use singular or plural names in the file name is yet another consideration to keep in mind for a professional presentation. You need to determine how to group the collection of files together as to give some visual representation that they are connected. Since FileMaker 7, developers could build all-in-one solutions where every table is self-contained within the same file. This is not always the best choice, for reasons beyond the scope of this document. With solutions being made up of multiple files, it is important from an administrative and evolutionary perspective to provide some convention that makes this very clear. Finally, you need to consider cross platform considerations as well as some general recommendations dealing with versioning.

I.1 Objectives:

- Recommended Character Set
- Recommended Case Convention
- Recommended Syntax Separation
- Recommended File Name Length
- Recommended Multi-file solution grouping
- Singular vs. Plural Guidelines
- Versioning Guidelines
- Extension Guidelines
- Issues with connectivity technologies (XML, ODBC, JDBC), including Reserved SQL words

I.2 Problem Definition:

Recommended Character Set: There are some basic restrictions on the characters FileMaker will allow. Additionally some characters can cause issues with external connectivity or data exchange in some RDBMS (Relational Database Management Systems). As a developer you need to be aware of the constraints of both FileMaker and any external system with which your solutions will interact.

Recommended Case Convention: This recommendation is not to overcome a specific problem but rather to make a consistent approach on how files are named. The objective is

to select a method and be consistent with it.

Recommended Syntax Separation: Syntax separation for file naming refers to the way one differentiates any prefix and or suffix from the file name. It is common to see prefixes or suffixes used to indicate a file is part of a group of files. They are also used to hold identification characteristics. The objective is to provide a consistent and universally recognized way to differentiate these leading/trailing meta-data elements from the file name itself.

Versioning Guidelines: File references are maintained by file name. To avoid issues with file references as versions change one should avoid using file names to represent the version of a solution.

Extension Guidelines: On Windows the file will always get the 'fpX' extension. On the Macintosh, the extension will default to using the extension but this optional. This becomes an issue when attempting to host a file without an extension from a Windows server, where upon starting the service looks for the .fpX extension to mount the file. The convention should address a consistent approach to an extension that will work on all supported platforms.

Singular vs. Plural Guidelines: The decision to utilize singular or collective nouns in place of plural alternatives is noted here only to suggest that as a developer you make a decision on which you will use and be consistent with it.

File Name Length Issues: The Operating System (OS) limits the file name length. However, it's not practical to have a 1000 character file name. However, one should consider a reasonable file name length and be aware that some applications that you may need to use will have various limitations. For example, a lengthy file name may have difficulty attaching to an email. If you need to interact with other applications, be aware that file name length could be an issue.

Multi-file Solution Grouping: In any multi-file solution there are primary file(s) and secondary files. A multi-file solution will have at least one primary file. The primary file(s) are 'exposed' to the end user. In a hosted solution these would be the file(s) the user is intended to select from the host dialog box. In a locally based solution these are the file(s) the user is intended to select directly from the OS or through a shortcut/alias. The support files are the subordinate files or those not 'exposed' to the user for direct access to a solution. In a hosted solution, these are the files that are used to support the function of the primary file. In addition, there are architectures that separate user interface from the data layer, in which case there are generally two separate files. Some designs place all of the files on the server, split between client and server, and even across multiple servers. The objective here is to recognize that files, wherever they are located, are part of the same solution or are connected in some way. There are a number of reasons this matters. From an administrative perspective it makes it much easier to manage and administer solutions for movement, security, and backups. From a development perspective it provides a consistent method that visually indicates a solution grouping.

External Connectivity: When interacting with any external RDBMS, you will need to be aware of any file name conflicts with either characters or reserved words. Additionally, when FileMaker files are accessed from other technologies, such as ODBC, JDBC, or XML you must specify the filename.

1.3 Standards Aware Guidelines (Files):

1. Should use only the characters.
 - Upper & Lower case aA – zZ
 - Number 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Should NOT contain spaces.
3. Should NOT start with numbers.
4. Should NOT use file name to indicate versions.
5. Should NOT contain periods other than the single period used for the extension.
6. Should consistently use singular or plural names.
7. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (File Names) | |
|--|--------------|
| lowerCamelCase | myFileName |
| UpperCamelCase | MyFileName |
| Single Underscore (lower Case) | my_file_name |
| Single Underscore (Title Case) | My_File_Name |
| Single Underscore (UPPER CASE) | MY_FILE_NAME |

8. Should contain the .fp7 (in lower case), regardless of platform. However, runtime solutions can use any extension; take care to not utilize other registered extensions.
9. Single-File Solutions are files that comprise all the tables for the entire solution. There are no additional recommendations.
10. Multi-File Solutions – Are a group of files that comprise all the tables for the entire solution. In any multi-file solution there are primary files and support files. The primary goal of multi-file solution grouping is to easily identify the files that comprise a solution. In many cases, natural names can be used. For example, consider a solution with 4 files that are named as follows:
 - Helpdesk.fp7
 - Tickets.fp7
 - Reports.fp7
 - Inventory.fp7

The naming does not clearly groups these files together. However, with some modifications it would be easy to do so using natural names. However, there are cases where natural names won't work. Or you may want to take a more consistent approach to your naming, regardless if a natural name works in some cases. Consider the same example using the standard aware recommendation. This aids in a few areas. First, it clearly identifies the primary file, secondly, it clearly identifies the support files, and lastly, it indicates the collection of files as a group.

- Helpdesk_hd.fp7
- hd__Tickets.fp7
- hd__Reports.fp7
- hd__Inventory.fp7

10.1.Support File(s) - Each support file should contain a prefix, defined here as the Logical Solution Identifier (LSI). The same LSI should be used for every support file in the solution. The LSI should follow the general conventions and should always be separated from the name with two underscores “__”. The use of two underscores aids in identifying the LSI from the file name, which could use a single underscore as a word separator. The LSI can be any length but should be consistent throughout your solution.

Syntax (Support Files)

<<LSI>>[__]myFileName[.fpX]

See [Syntax Legend](#) for description of syntax.

- LSI (Logical Solution Identifier) – required, developer defined
Denotes files are logically connected
- “__” – required
Separates LSI from File Name
- myFileName –
Descriptive File Name
- .fpX – required
All files should contain the “.” (period) followed by fpX extension, where X is file format version of FileMaker. Example : .fp7

10.2.Primary File(s) - You may elect to include the LSI in the primary file(s) as a prefix or suffix. In either case the LSI should always be separated from the name with two underscores “__”.

Syntax (Primary Files)

myFileName[__]{LSI}[.fpX]

{LSI}[__]myFileName[.fpX]

See [Syntax Legend](#) (Appendix D) for description of syntax.

- LSI “Prefix” (Logical Solution Identifier) – required, developer defined
Denotes files are logically connected
- “__” – required
Separates LSI from File Name
- myFileName –
Descriptive File Name
- LSI “Suffix” (Logical Solution Identifier) – required, developer defined
Denotes files are logically connected
- .fpX – required
All files should contain the “.” (period) followed by fpX extension, where X is file format version of FileMaker. Example : .fp7

10.3.Examples

| | |
|-------------------------|----------------|
| MySolution__xxx.fp7 | - Primary File |
| xxx__MySupportFileA.fp7 | - Support File |
| xxx__MySupportFileB.fp7 | - Support File |
| xxxx__mySolution.fp7 | - Primary File |
| xxxx__supportFileA.fp7 | - Support File |
| xxxx__supportFileB.fp7 | - Support File |
| My_Solution.fp7 | - Primary File |
| ABC__my_solution.fp7 | - Support File |
| ABC__support_file_b.fp7 | - Support File |

1.4 Ancillary Considerations:

- Borrowed Files: In some cases a solution may include a reference to another file that is not individually associated with the specific solution. For example, an employee directory might be used across many solutions. In these cases it is recommended to treat the 'borrowed' file, as its own solution and not incorporate it as part of any specific solution.
- Client File Storage: For clarification and organization you should consider placing support files residing on a local drive in a sub-folder/directory under the primary file location. In solutions that spread files on both the server and client, consider creating a program folder and housing your client side primary files at the root of this folder with all support files located in a sub-folder

2 Table Naming

Table names are not exposed to any major degree within the development environment. Interacting directly with tables is generally limited to associating a table with a Table Occurrence, found on the Relationship Graph. All data interactions are through the Table Occurrence. It is important to make this distinction. While many of the dialog boxes will use the term 'Table' you will actually be utilizing the 'Table Occurrence'.

When selecting table names there are a number of factors to consider. As with all areas you need to select a character set that will not encumber connectivity to the solutions from technologies such as ODBC, JDBC, and XML. For the actual tables themselves this is not directly important, due to the fact you will not reference tables directly. However, if you choose to use the Source Table name in your Table Occurrence name, which is utilized for connectivity calls and referenced through all interactions, you will need to be more selective. Obviously, not every solution will utilize external connectivity, but as solutions evolve it is best to be prepared and plan for the possibility.

For consistency and professionalism you need to select a case and be consistent with it. You will need to select how the individual words within your table name will be separated. Consideration must be made to how the table name itself will be syntactically separated from any meta-data, prefix or suffix. Syntax separation for table names might be used if one chooses to provide some functional or categorization within the name. For example, indicating within the name a 'join' or 'session' table by using a prefix such as 'j__TableName' or 'ses__table_name'. In both cases the syntax separator is a "__" double underscores. Be consistent with the use of singular or plural names. Consider how some use of 'white-space' can help add some organization to the Table tab. This can be very helpful when dealing with large numbers of tables. Finally, while a table name can be 100 characters, this is not practical for a number of reasons.

2.1 Objectives:

- Recommended Character Set
- Recommended Case Convention
- Plural vs. Singular Recommendations
- Table Name Length Recommendations
- Options for white space to address lack of organization
- Guidance for Table Type Naming (eg Join, Utility, System, Reference)

2.2 Problem Definition:

Recommended Character Set – Table names are somewhat protected from any character related issues due to their limited usage throughout the application. However, some consideration should be given to the characters used assuming that one may include the table name in other areas, such as Table Occurrence names, where they are referenced.

Recommended Case Convention – Selecting a convention to use for Table Names must be made with respect to how you will name Table Occurrences and Layouts. The decision on whether to use UpperCamelCase, lowerCamelCase, or UPPERCASE for the most part is

simply a preference. However, one must consider a few factors. First, will you want to include the Table Name in the Table Occurrence name? If so, you must consider the convention you want to use for Table Occurrence names. Assuming you want to use all UPPERCASE, you need to determine how you will separate words (a space, underscore, double underscores etc). In addition, if you use underscores to separate words then what will be used to separate any syntax you assign in Table Occurrences, Layouts, and other areas where you need to include the Table Name. It's important to make your selections for any section with respect to where and how it will be referenced elsewhere.

Table Name Length Issues – A Table Name can be up to 100 characters in length. This however, is not a practical limit in many cases. Referring back to Recommended Case Convention, the Table Name might be included in the Table Occurrence Name, thus this reference to the Source Table Name in the Table Occurrence name will add length to the name of the Table Occurrence. Adding to this any additional convention employed at the Table Occurrence can create a Table Occurrence name that may be longer than some interface dialogs will display without some manipulation. The issue is raised here to emphasize that putting some thought into the length of your Table Names must be made in respect to other areas where conventions will be applied.

Singular vs. Plural Guidelines: The decision to utilize singular or collective nouns in place of plural alternatives is noted here only to suggest that as a developer you make a decision on which you will use and be consistent with it.

Table Categorization – Some developers like to include meta-data in the table name. In most cases this categorizes the table as holding a specific type of data or indicating it performs a specific role in the solution structure. A few examples are tables constructed for sessions, reference, utility, and joins. Within FileMaker there is no capability to denote or comment this type of categorization. Therefore, various developers include this meta-data as part of the table name.

Spacing Tables – There are some practices amongst the community that utilize a technique that creates “Spacing Tables” these tables hold no fields but are used to act a separators between categories of table. Smaller solutions will not benefit from this technique, but as solutions grow and include larger amounts of tables it can be very helpful.

2.3 Standards Aware Guidelines (Tables):

1. Should contain only the characters
 - Upper & lower case aA-zZ
 - Numbers 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Should NOT contain spaces
3. Should consistently use singular or plural table names.
4. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Table Names) | |
|---|---------------|
| lowerCamelCase | myTableName |
| UpperCamelCase | MyTableName |
| Single Underscore (lower Case) | my_table_name |
| Single Underscore (Title Case) | My_Table_Name |
| Single Underscore (UPPER CASE) | MY_TABLE_NAME |

5. Should NOT use any FileMaker Reserved Words
6. Should NOT use any SQL Reserved words. See Appendix A. Additionally check the documentation of the RDBMS.
7. Table Categorization - As mentioned in the problem definition, some developers like to indicate the ‘type’ of table or the function of the table within the table name itself. If your development practice needs to include this we recommend the following syntax.
8. Syntax: {function}[__]TableName
See [Syntax Legend](#) (Appendix D) for description of syntax.
 - function – optional; developer defined
Provides a developer defined categorization for extending as necessary. As the developer you are able to define the various categories. Be considerate of case, length, and consistency when defining. You should document your selections following the [Adherence section](#) guidelines in this document.
 - “__” – required
A double underscores should be used to indicate the separation of function and Table Name.
 - TableName –
The descriptive name for your table.

2.4 Ancillary Considerations:

Usage of Spacing Tables: In solutions with a large number of tables it can be useful to add some categorization to your table name list. Because, the development environment does not provide this capability it’s up to the developer to devise a way to accomplish this. In the example below we have a relatively small list of tables, however, it will do for our example. Consider a much larger list and the value becomes more apparent. We have created tables with some indication they are acting as headers for a category. The example shows Accounting, Communications, and ‘-Registration-’. These are simply tables that have no fields. You may want to remove them from

the Relationship Graph, since they will never be used in your solution. But, as you can image, it might make things a little more organized in those larger solutions. In our example we are using a “-“ at the front and back. You can use whatever works for you. The point is to use something that stands out for you. Be aware that FileMaker will give you a warning about certain characters, but remember you’re not going to use these tables in any way.

| Table Name |
|--------------------|
| * ----- |
| * -Accounting- |
| * Customers |
| * Enrollments |
| * Invoices |
| * InvoiceItem |
| * Products |
| * -Communications- |
| * Events |
| * Parents |
| * Supplies |
| * -Registration- |
| * Attendance |
| * Classes |
| * Employees |
| * Students |

Figure 1

3 Field Naming

Field names are an area where the topic of conventions is widely varied. Generally, there are two categories of thought. Those who want to use “natural” names and those who utilize some form of notation to expose some meta-data about the field. This notation is used to expose information about the field, which is unavailable outside of the Define Database dialog. There are many variants to the detail the notation takes. But the underlying idea is the same in all.

When selecting field names there are a number of factors to consider. As with all areas, one needs to select a character set that will not encumber connectivity to the solutions from technologies such as ODBC, JDBC, and XML. For consistency and professionalism you need to select a case and be consistent with it. You will need to select how the individual words within your field name will be separated. Again, consistency and professionalism are the objective.

Consideration must be made to how the field name itself will be syntactically separated from any meta-data, prefix or suffix. The syntax separation for field names becomes very important due to the variety of problem we are trying to address. One needs to consider field categorization, indicating key/match fields, utility fields or developer fields. The objective is to clearly be able to distinguish what is notation and what is not, without having to decipher code. This being said, if you choose to use notation then there needs to be a universal understanding of this notation. Therefore, the FDC provides two basic components to assist with this. First the convention provides a common syntax that provides the ‘location’, where to put the parts. The most common components should be noted and provided. And finally, within the syntax provide the ability to ‘extend’ or customize it to a developer’s more specific needs.

Consideration must be given to where and why the components of the syntax are placed in specific order of recommendation. You need to utilize singular or plural names and be consistent. With field names it is important to look at ways to do field grouping, such as using NameFirst, NameLast as opposed to FirstName and LastName. One of the issues considered was to utilize the field name sort, rather than a custom sort order. While this is certainly not a primary decision for how to name your fields, utilizing this built-in capability can reduce time spent in organization activities. You should consider how some use of ‘white-space’ could help add some organization to your field lists. This can be very helpful when dealing with large numbers of fields, especially if you prefer a manual sorting method for field listing.

The FDC realizes that your development practices may not fit with what is outlined. It really comes down to a choice at this point. If you’re going to use some notation, then give the recommendation serious consideration.

3.1 Objectives:

- Recommended Character Set
- Recommended Case Convention
- Field Name Length Issues
- Field-naming options for lack of field meta-data in the development environment.
- Limitations that prevent the ‘hiding’ of utility fields from user interface which addresses limitation of development environment.
- Field and meta-data separation
- Options for white space usage.
- Options for field grouping and readability

3.2 Problem Definition:

Recommended Character Set: The usage of improper characters can cause difficulties in a number of areas. Most important are the usage of valid characters within calculations. Developers should take care to utilize characters that do not conflict with FileMaker calculations, avoid issues with variables, and be aware of any characters that may present a problem with exchanging data with Relational Database Management Systems (RDBMS).

Recommended Case Convention: The decision of how to separate words within field names has an impact of overall consistency throughout conventions. For example, choosing to separate individual words with an “_” (underscore) will make it difficult to determine the field name from any prefix or suffix that one might choose to utilize for the representation of field meta-data. One of the most common uses of this is the use of “g_” to precede the field name.

Field Name Length: A FileMaker field can be up to 100 characters in length. This however, is not a practical limit in many cases. With the release of version 8, most dialog boxes will have no difficulty in displaying the full name. However, Sort, Export, and Edit Find Request are limited. Sort is limited to 20 characters on OS X and 26 characters on Windows. Export is limited to 30 on OS X and 28 characters on Windows. Edit Find Requests is limited to 26 characters on OS X and 27 on Windows. These numbers vary depending on the width of the characters. For example “8” is wider than “1”. The numbers provided are based on a string of full width characters. The FDC does not make any recommendations on length but provides these limitss for reference.

Lack of field meta-data within development environment: The field is one of the primary programmable objects with which a developer works. Setting aside a remarkable memory, it is difficult, if not impossible to know the meta-data of a field without opening Define Database. What is its data type? What is dependent on it? How does it store data? How will it behave in a script? This information is not readily available. Whether you agree that this is a challenge or not determines how you will approach it with a naming convention. Assuming, one wants to expose this information at the field name level, considerations on how to accomplish this ought to be part of the convention.

Inability to hide utility fields: “Utility” fields are fields defined to store transitory data or information that is not generally exposed to the end user. In any case, the challenge to the developer is that they cannot prevent end users from seeing these fields in all fields presenting dialog boxes. While FileMaker 8 provides the functionality to list only the fields on a layout for exporting and sorting to some degree, there are many other areas where all the fields will be listed. In this case the inability to hide the fields creates a situation where a developer may want to list these fields out-of-the-way. In most cases this ends up being the bottom of the field list. Since version 7, FileMaker uses Unicode sort order for field names. Other than manually sorting field names, which can be cumbersome and time consuming, the only option is to establish a naming convention, which moves utility fields to the bottom of the list. Due to Unicode sorting order the character “z”, “zz”, or a Unicode character that sorts these field to the bottom is generally agreed to be the best way to ensure a field is listed last. As a developer you must decide if this is important to you or not.

Field and meta-data separation syntax: There needs to be a clear understanding for the current and future developers as well as the end-user on what separates the field name from the meta-data/notation. In addition, there needs to be a clear and consistent designation between what is notation and what is the field name.

3.3 Standards Aware Guidelines (Fields):

The FDC has elected to recognize three (3) broad categories for fields. These are General Fields, Key Fields, and Utility Fields.

3.3.1 General Fields

A General Field is categorized as those fields used for general data storage.

1. Should contain only the characters
 - Upper & lower case: a-z or A-Z
 - Numbers: 0,1,2,3,4,5,6,7,8,9
 - Single and Double underscores “_” “__”
2. Should NOT contain spaces
3. Should NOT start with numbers
4. Should be consistent with usage of singular or plural names
5. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Privilege Sets) | |
|--|---------------|
| lowerCamelCase | myFieldName |
| UpperCamelCase | MyFieldName |
| Single Underscore (lower Case) | my_field_name |
| Single Underscore (Title Case) | My_Field_Name |
| Single Underscore (UPPER CASE) | MY_FIELD_NAME |

6. Should sort field list by field name
7. Depending on the character or characters you use as your “Utility Field” indicator you should reserve this character by not starting any field with that letter. In the utility fields area the letter “z” and “zz” as well as other Unicode characters that will put utility fields to the bottom of the field list, assuming fields are sorted by name.
8. Should use Field Grouping when possible. Examples include:
 - Name_First
 - Name_Last
 - AddressCity
 - AddressState
 - AddressPostalCode
 - passportNumber
 - passportIssuingCountry
 - passportExpirationDate
 - passportName
9. Derived or Calculated Fields determined by the developer to not exist as “Utility” fields but want to include notation should follow the recommendations for “Utility” fields as a suffix.

For example, InvoiceTotal is calculated, however, it is not a field you may want to ‘hide’ from the user. However, you may want to include some notation. In such a case you would name the field InvoiceTotal__lcn. Indicating the field is locally stored, calculated, of number type result. See [utility naming](#) for additional information.

3.3.2 Key/Match Fields

FileMaker does not enforce or mandate many of the relational integrity constraints present in SQL-based, Relational Database Management Systems. For Example, it does not force you to create a primary key for a table. There is no indication, visually or contextually, within the application that a field is a key/match field. In most cases it is desirable or necessary to utilize keys or match fields within your solutions. You can utilize some notation to specifically address identification of key fields.

Should you choose to utilize some notation for key fields, the following syntax is the recommendation for the standards aware approach.

Key/Match Field Syntax Directives:

- Provide a universally available, understood, and consistent map to key meta-data notation.
 - Identify Key/Match field
 - Identify Function of the Key/Match field
 - Identify Key/Match field storage
 - Identify Key/Match field data type
 - Sort the Key/Match fields to the top of the fields list while maintaining a field sort by name (Alphabetical Sort Order). This reduces manual sorting by developers and presents expected order by users of the solution.
 - Utilize notation characters that are either generally acceptable (popular), such as “p” for primary key and “f” for foreign key, or align with the FileMaker development environment.
1. Should contain only the characters
 - Upper & Lower Case: a-z or A-Z
 - Numeric: 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
 2. Should NOT contain spaces
 3. Should NOT start with a number
 4. Should be consistent with usage of singular or plural names
 5. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Field Names) | |
|---|---------------|
| lowerCamelCase | myFieldName |
| UpperCamelCase | MyFieldName |
| Single Underscore (lower Case) | my_field_name |
| Single Underscore (Title Case) | My_Field_Name |
| Single Underscore (UPPER CASE) | MY_FIELD_NAME |

6. Syntax: [k]<function>(storage)(type)[__]DescriptiveName

See [Syntax Legend](#) for description of syntax.

- “ ” – required
Single underscore will place keys at the top in all dialogs where fields are listed and the relationship graph.
- k - lower case; required
Denotes the field as a key/Match Field
- function - lower case; required; uses provided values if used
Denotes key field category and/or function it will serve.
 - p – Primary Key
 - f – Foreign Key
 - a – Alternate Key
 - c – Compound / Concatenated / Calculated Key
 - m – Multi-Line Key
- storage - lower case; optional; uses provided values if used
Denotes field storage
 - l = Locally Stored (lower case “L”)
 - g = Globally Stored
- type - lower case; optional; uses provided values if used
Denotes field type
 - t = Text
 - n = Number
 - d = Date
 - i = Time
 - m = Time Stamp
- “ ” – required
Double underscore should be used to separate the key syntax from the Descriptive Name for the field.
- DescriptiveName- Selected Case
Key Field Descriptive Name

▪ Examples

| Full format using all optional meta-data | Minimal format using no optional meta-data |
|--|---|
| <u>kp</u> <u>lt</u> <u> </u> InvoiceID | <u>kp</u> <u> </u> InvoiceID |
| <u>kc</u> <u>gt</u> <u> </u> SelectedParticipants | <u>kc</u> <u> </u> SelectedParticipants |
| <u>kft</u> <u> </u> CustomerId | <u>kf</u> <u> </u> CustomerId |
| <u>km</u> <u>gt</u> <u> </u> SelectedDaysView | <u>km</u> <u> </u> SelectedDaysView |
| <u>kft</u> <u> </u> Invoice <u> </u> line <u> </u> item | <u>kf</u> <u> </u> Invoice <u> </u> line <u> </u> item |

3.3.3 Utility Fields

The Utility field naming is designed to create a consistent way to accommodate the various needs of developers and address some of the most common issues with field naming for those fields that are generally not desirable to make available to the end user.

Utility Field Directives:

- Provide a universally available, understood, and consistent map to utility field meta-data notation.
 - Identification of Utility field
 - Identification of function of a Utility field that is customizable
 - Identify Utility field storage
 - Identify Utility field data type
 - Identify Utility field as a repeating field
 - Sort utility fields to the bottom of the fields list while maintaining a field sort by name (Alphabetical Sort Order). This reduces manual sorting by developers and places these fields where they are out of the way for the user.
1. Should contain only the characters
 - Upper & Lower Case: a-z/A-Z
 - Numeric: 0,1,2,3,4,5,6,7,8,9
 - Single and Double underscores “_” “__”
 2. Should NOT contain spaces
 3. Should NOT start with numbers
 4. Should be consistent with usage of Singular or Plural names
 5. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Privilege Sets) | |
|--|---------------|
| lowerCamelCase | myFieldName |
| UpperCamelCase | MyFieldName |
| Single Underscore (lower Case) | my_field_name |
| Single Underscore (Title Case) | My_Field_Name |
| Single Underscore (UPPER CASE) | MY_FIELD_NAME |

6. Syntax: <zz>{function}[__]DescriptiveName[__](storage)(type)(repetition)
See [Syntax Legend](#) for description of syntax.
 - “zz” or Unicode character that sorts to bottom – required
Indicates the field as a developer key and sorts to bottom of all dialog boxes where field names are present.
 - function – lowercase; optional; developer defined
Provides a developer-defined categorization for extending as necessary. As the developer you are able to define the various categories. Be considerate of case, length, and consistency when defining. You should document your selections following the [Adherence section](#) guidelines in this document.
 - “__” – required
Double underscores denotes separation of prefix from Descriptive Name

- **DescriptiveName**
Developer selected desired name. Follows general field guidelines.
- **“__” – required**
Double underscores will be used to indicate the end of the DescriptiveName and beginning of suffix notation.
- **storage - lower case; required; uses provided values if used**
Used to denote the field storage
 - l = Locally Stored (lowercase L)
 - g = Globally Stored
- **type - lower case; required; uses provided values if used**
Indicates the data type used or returned.
 - **Non-Calculated Result Types**
 - xt = Text
 - xn = Number
 - xd = Date
 - xi = Time
 - xm = Time Stamp
 - xr = Container
 - **Calculated Result Types**
 - ct = Text
 - cn = Number
 - cd = Date
 - ci = Time
 - cm = Time Stamp
 - cr = Container
 - **Summary Result Type**
 - xs = Summary
 - **repetition - lower case; required; uses provided values if used**
Indicates the data is stored with repetitions
 - p = repetitions

3.3.4 Field Notation Reference

| Field Notation Reference | | | | | | |
|--|-------|--------|-------|--------|------------|-----------|
| | Text | Number | Date | Time | Time Stamp | Container |
| Keys (long form: Using all meta-data) | | | | | | |
| Key:Primary | _kplt | _kpln | _kpld | _kpli | _kplm | N/A |
| Key:Foreign | _kflt | _kfln | _kfld | _kfli | _kflm | N/A |
| Key:Alternate | _kalt | _kaln | _kald | _kali | _kalm | N/A |
| Key:Compound | _kclt | _kcln | _kcld | _kclic | _kclm | N/A |
| Key:Multiline | _kmlt | _kmln | _kmld | _kmli | _kmlm | N/A |
| Repeating Keys (Add “p” to end of notation, | p | p | p | p | p | N/A |

| | | | | | | |
|---|-------|-------|-------|-------|-------|-------|
| in lowercase) | | | | | | |
| Utility Fields / General & Calculated Fields (Full form: Using all meta-data) | | | | | | |
| General (local Storage) | __lxt | __lxn | __lxd | __lxi | __lxm | __lxr |
| General (Global Storage) | __gxt | __gxn | __gxd | __gxi | __gxm | __gxr |
| Calculated (local Storage) | __lct | __lcn | __lcd | __lci | __lcm | __lcr |
| Calculated (Global Storage) | __gct | __gcn | __gcd | __gci | __gcm | __gcr |
| Repeating (Add “p” to end of notation, in lowercase) | p | p | p | p | p | p |
| Summary (Local Storage) | N/A | __lxs | N/A | N/A | N/A | N/A |

3.4 Ancillary Considerations

Field Separators: Many developers utilize a ‘non-functional’ field to act as a separator in the field list. Spacing in long lists of items makes them more manageable. It allows for some kind of division, where the development environment lacks this capability. When utilizing this technique it is recommended to use a field type of number and set it for global storage to reduce storage size.

4 Table Occurrences

Since FileMaker 7 and the advent of the Relationships Graph, the product has extended its capabilities to better support large and complex systems. However, the Relationships Graph can become cumbersome and unwieldy when representing complex applications. For example, a complex system may have the same Source table represented multiple times on the graph. Couple this with the requirement that there can be at most one relational path between any two Table Occurrences. The graph can become large and challenging to navigate. While the Relationships Graph gives the developer a Graphical User Interface (GUI) for working with relationships, it does not offer the ability to logically group or name groups of Table Occurrences. This makes it difficult to know which Table Occurrence to work with when selecting from a pop-up list. All Table Occurrences are listed in alphabetical order in the list and lack any means of logical grouping. There are numerous places where the need to select a Table Occurrence exists, such as placing fields on a layout or selecting a field in the calculation dialog. In these situations, there is no way to see a Table Occurrence's underlying source table or its context in the Relationships Graph. This limitation prompts us to look for a naming convention that will embed the source and contextual description in the Table Occurrence name. The proposed naming conventions are for indicating source AND context.

It is very clear that any attempt to define a naming convention will be closely tied to an individual developer's methodology for representing Table Occurrences. Thus, the FDC has elected to illustrate several methods for building Relationship Graphs, and their supporting naming conventions, as a resource to draw upon for your solutions, rather than propose a single convention that might favor a particular development method.

Ultimately, regardless of the specific method used, a clear and consistent understanding of the specifics of your solution should be recognizable to another developer with a minimal amount of effort. Leveraging a generalized approach that can be further refined is a solid first step. In each of the methods examined you will find many similarities. They all share a similar format. Each is grouped with a prefix. Following this is the Source Table Name. Lastly there is an optional Descriptive Name where additional context in the name is beneficial.

4.1 Objectives:

- Terminology
 - Define Table
 - Define Table Occurrence
 - Define Source Table/Base Table
 - Define Functional Table Occurrence Group (FTOG)
 - Define Primary Table Occurrence (PTO)
- Recommended Character Set
- Recommended Case Convention
- Table Occurrence Length issues
- Provide various mechanisms to address the challenge of relationship graph context when selecting a Table Occurrence while working with calculations, adding fields to a layout, or defining layout context.
- Provide various mechanisms to address the difficulty in grouping Table Occurrences for the purpose of exposure to the development environment.

- Provide a method to address the dependency on a Primary Table Occurrence of calculations that are self-contained.
- Provide various methods to overcome organizational challenges of Table Occurrences.
- Issues with connectivity technologies (XML, ODBC, JDBC), including reserved SQL words.

4.2 Problem Description:

Definitions: For the purposes of this document and a universal understanding of terms we provide the following definitions.

- **Table** - A collection of data pertaining to an entity, such as customers or stock prices. A database file contains one or more tables, which consist of fields and records. When you create a new table, a visual representation, or Table Occurrence, of the table appears in the Relationships Graph. You can specify multiple occurrences (with unique names) of the same table in order to work with complex relationships in the graph.
- **Table Occurrence:** A Table Occurrence refers to an instance of a table on the Relationships Graph. Keep in mind that all interactions with a table throughout the development environment will interact with Table Occurrences. This is the one and only way to 'address' a table and its contents.
- **Source Table:** A Table Occurrence is associated with a Table. The Source Table refers to the Table with which the Table Occurrence is associated. In the example below, a Table Occurrence named "AdmissionInterface|Classes|ClassList" is associated with a Source Table named Classes. The Source Table will also be referred to as the Source Table Name.

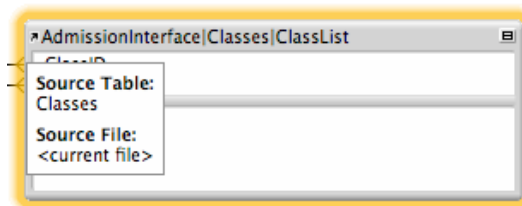


Figure 2

- **Primary Table Occurrence:** PTO is a name for a special Table Occurrence. The PTO serves as the designated Table Occurrence that will be used when creating calculations that are 'internally referenced', those calculations that are derived from data contained solely in the same table, not making use of related data.
- **Spacing Table:** A name for a technique used to create 'label' or 'separator' tables within the Define Database dialog. This technique uses tables that contain no fields for the purpose of grouping and categorizing tables and Table Occurrences.

Recommended Character Set: There are some basic restrictions on the characters FileMaker will allow. Additionally some characters can cause issues with external

connectivity or data exchange with some RDBMS (Relational Database Management Systems). As a developer you need to be aware of the constraints of both FileMaker and any external system with which your solutions will interact.

Recommended Case Convention: This recommendation does not overcome a specific problem but rather encourages a consistent approach in how Table Occurrences are named. The objective is to select a method and be consistent with it.

Recommended Syntax Separation: Syntax separation for Table Occurrence naming refers to the way one differentiates any prefix and or suffix (meta-data) from the remainder of the Table Occurrence name. It is common to see prefixes or suffixes used to indicate certain components of the name. They are also used to hold identification characteristics. The objective is to provide a consistent and universally recognized way to differentiate these leading/trailing meta-data elements.

Lack of context to relationship graph when selecting a Table Occurrence while working with calculations, adding fields to a layout, or defining layout context: Outside of the Relationships Graph, for example when selecting the Table Occurrence (referred to as “TO” or “TOs”) from a list, you are presented with a long list of TO names to use. This list lacks any grouping or order unless your naming convention supplies the context.

Dependency on a Primary Table Occurrence on calculations that are self-contained: When creating calculations you must select the context from which the calculation will evaluate. For fields where the calculation result is derived without referencing related data, there is some consideration that these calculations should always be based on the same context.

Lack of Table Occurrence Organization Capabilities: Regardless of what method is used to create the Relationship Graph, it is generally agreed that a number of TOs often work together to provide some function, irrespective of the connectivity to other TOs on the graph. With this understanding it only makes sense that one would want to organize this group in some way. This capability does not exist unless your methodology and naming convention provides it.

4.3 Standards Aware General Guidelines (Table Occurrences):

1. Should use only the characters
 - Upper & lower case: aA or zZ
 - Numbers: 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Does NOT contain spaces
3. Does NOT start with a number
4. Does NOT contain periods
5. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Table Occurrences) | |
|---|--------------------------|
| lowerCamelCase | myTableOccurrenceName |
| UpperCamelCase | MyTableOccurrenceName |
| Single Underscore (lower Case) | my_table_occurrence_name |
| Single Underscore (Title Case) | My_Table_Occurrence_Name |
| Single Underscore (UPPER CASE) | MY_TABLE_OCCURRENCE_NAME |

4.3.1 Functional Spider Grouping (FSG) Method

This first approach to organizing table occurrences is to name each with a prefix indicating its function. The image of the graph once this method is in place is much like a spider or web and thus the term spider is used to represent this approach. Note that this is not the only method of designing functional groupings (see the following example of Functional Table Occurrence Groups).

The main idea with the functional grouping is that you are not representing table occurrences by name or by relationship but rather more by functional groupings. The types of functions that can exist are unlimited and will depend on the demands of your solution and your implementation. (scripts, value lists, portals, etc.)

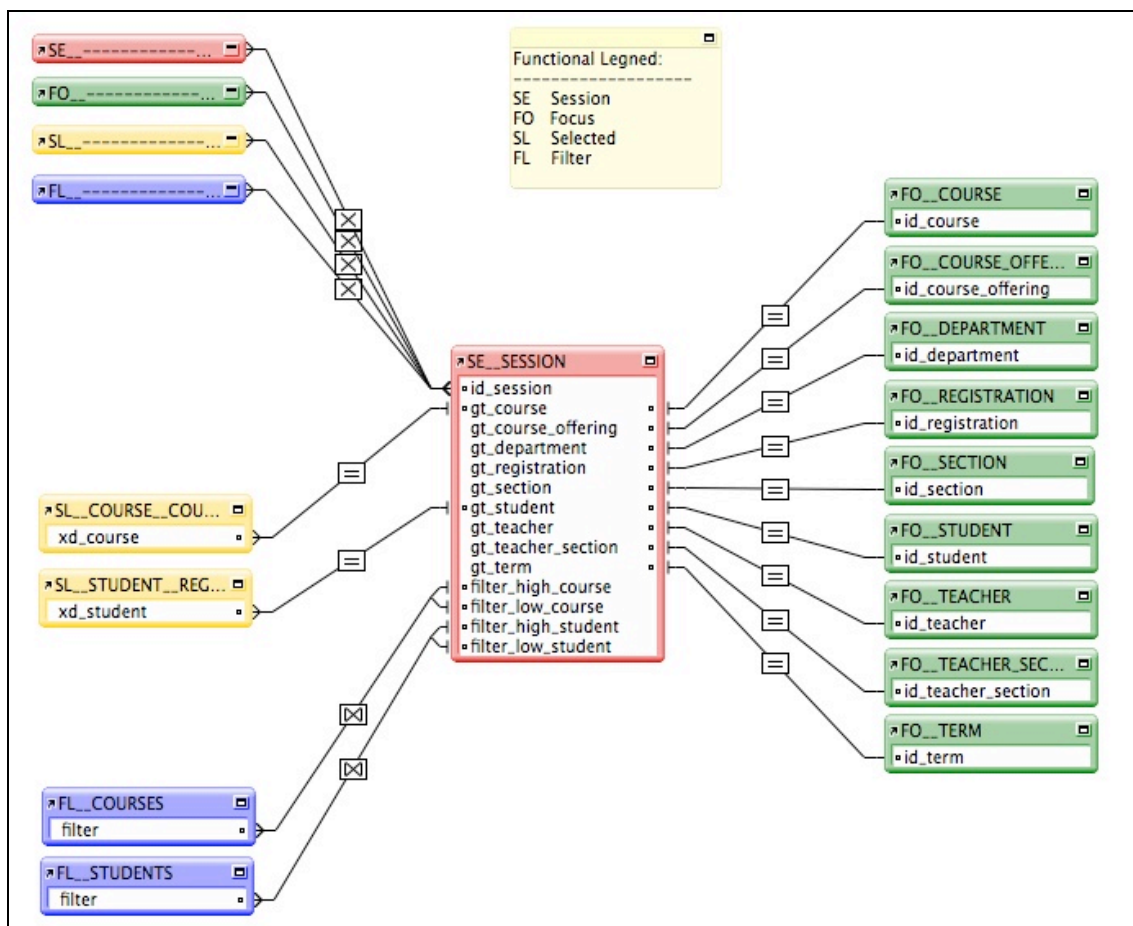


Figure 3

In this method, functional groups are identified with a “functional prefix”. In this case it is a two-letter prefix. Color can also be used to help locate the different functional groups. And new to FileMaker 8 is the ability to also attach notes to the graph. In this case we added a note to describe what each of the two-letter functional groups represents.

The intent here is to not describe in too much detail how this Functional Spider Group (FSG) method works but rather to introduce the main concepts.

One of the functional groups in the graph is represented with a two-letter prefix labeled FO, which we will use for the word “Focus”. All the tables in the graph that have this two-letter prefix will be found grouped together when you need to enter information related to the record that has focus.

Because there would be so many tables in the graph with this approach, a separator table is used to also provide a way for each of these functional groupings to show up all together. These spacing tables also share the same two-letter prefix and when connected to the FSG they allow for a separation and grouping to help in locating the function you want to work with.

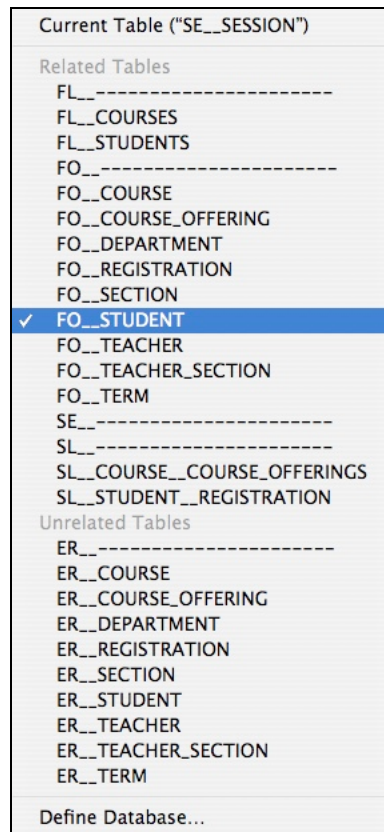


Figure 4

The above is an example of selecting a field when on a session layout. The functional separators make it easy to locate a particular functional group in the popup list of Table Occurrences.

4.3.1.1 Functional Spider Grouping Pros

- Reduces number of Table Occurrences
- Provides organization by function
- Works well for small-medium sized solutions. (Most solutions are in this category.)

- Can work with large solutions with the caveat that it's dependent upon the number of functional needs. The more functional needs the system must serve, the less attractive this organizational method becomes.
- Supports bi-directional relational model by allowing layouts to use any TOG within the FSG. This reduces the number of TOs needed by not restricting layout exposure to only one TO in the FSG.
- Provides functional grouping within Table Occurrence menus outside of the Relationships Graph in a reasonably grouped order.

4.3.1.2 Functional Spider Grouping Cons

- More attractive for “portal-driven” methods. Does not work well with more “open” solutions.
- Naming does not attempt to expose structure outside of the graph.
- More difficult to locate the specific TOs on the Relationships Graph due to the ability to base a layout on any TO within the FSG.

4.3.1.3 Functional Spider Grouping: Standards Aware Guidelines

- I. Syntax: <<FunctionalPrefix>>[__]DescriptiveName
See [Syntax Legend](#) for description of syntax.
 - FunctionalPrefix – required; developer defined
Provides the mechanism to represent a logical separation of interconnected Table Occurrences. This could be an abbreviation or functional name.
 - DescriptiveName -
Provides the mechanism to give meaning to the Table Occurrence.
 - “__” - required
The usage of a “__” double underscore is the recommended separator character. This allows the usage of underscores within Source Table names and Logical Table Occurrence Names while still providing readability and parsing capabilities.

4.3.2 Functional Table Occurrence Groups (FTOG) Method

Another approach is “Functional Table Occurrence Grouping” (FTOG), where the Relationship Graph has multiple ‘mini’ functional graphs, each one consisting of only the Table Occurrences that provide the functionality for that cluster or FTOG. One of the key differentiators for the FTOG method is that it breaks down the various components or functions of a solution into smaller sub-sets. This provides, in some cases, a more manageable graph. It also reduces the number of Table Occurrences to search within the Relationship Graph. For any given function you need only refer to the associated FTOG group. Each FTOG will have only those Table Occurrences relevant to the functionality required.

With any variant of the FTOG method one of the objectives is to provide a meaningful name to what purpose this group or cluster of Table Occurrences is providing. The meaning of the grouping can be varied but the importance is that the group or cluster provides functionality and needs to have a name associated with this functionality so that it may be understood or at least recognized outside of the Relationship Graph. Figure 2 shows a Relationships Graph organized using the FTOG method. Each of these FTOGs represents some functionality that is independent from the other FTOG groups. Using The FileMaker 8 text tool we can place notes around each Functional Group for some additional in-graph understanding of the FTOG.

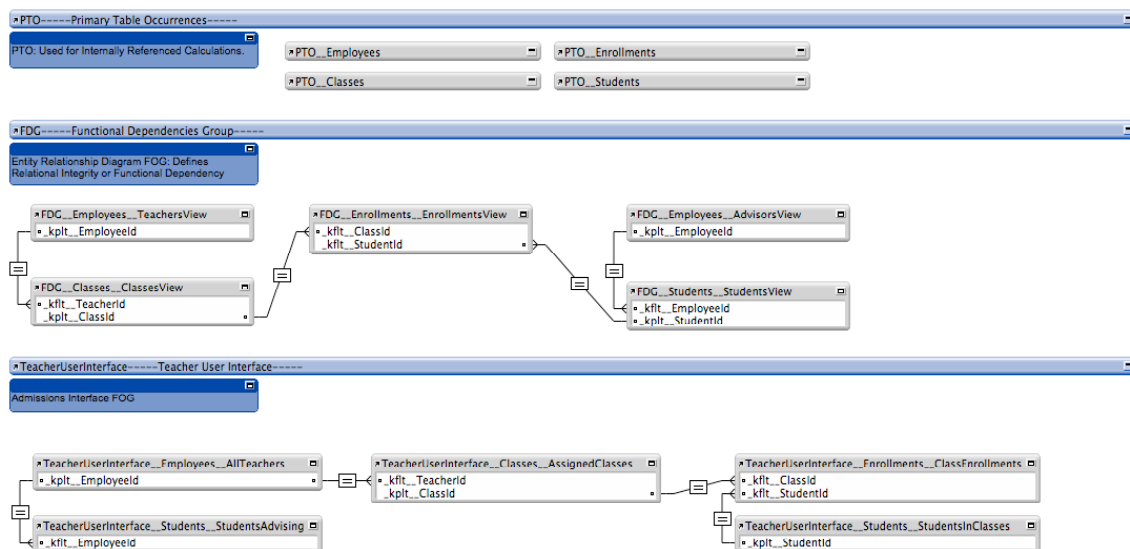


Figure 5

4.3.2.1 Functional Table Occurrence Grouping (FTOG) Pros

- Reduces complexity of Relationship Graph
- Reduces number of Table Occurrences
- Provides visual and list organization by functionality
- Works well for small - medium sized solutions. (ost solutions are in this category.)
- Can work with large solutions with the caveat that it's dependent upon the number of functional needs. The more functional needs the less attractive it becomes.
- Related tables within a FTOG are fairly limited, and they are grouped into sorted alphabetical order within the FTOG.

- Supports bi-directional relational model by allowing layouts to use any TOG within the FTOG. This reduces the number of TOs needed by not restricting layout exposure to only one TO in the FTOG.
- Supports a wide range of development methodologies: those under highly scripted “portal-driven” solutions and more open based solutions using FileMaker native controls.
- Provides functional grouping to Table Occurrence menus outside of the Relationships Graph in a reasonably grouped order.

4.3.2.2 Functional Table Occurrence Grouping (FTOG) Cons

- Naming is considered to be more stringent.
- Naming does not attempt to expose structure or meaning outside of the graph.
- More difficult to locate the specific TO on the Relationships Graph due to the ability to base a layout on any TO within the FTOG.

4.3.2.3 Functional Table Occurrence Groups: Standards Aware Guidelines

I. Syntax:

<<FunctionalTableOccurrenceGroup>>[__]<SourceTableName>[__]DescriptiveName

See [Syntax Legend](#) for description of syntax.

- FunctionalTableOccurrenceGroup (FTOG) – required; developer defined
Provides the mechanism to name a collection of Table Occurrences. This could be an abbreviation or functional name. For example, one could use FD at the front of the collection of Table Occurrences that represent the Functional Dependency group. Another example may be ‘UserInterface’ to indicate the collection of Table Occurrences that depict the group which provides the interface.
- “__” – required
The usage of a “__” double underscore is the recommended character. This allows the usage of underscores within Source Table names and Logical Table Occurrence Names while still providing readability and parsing capabilities.
- SourceTableName – required, uses source table name
Includes the Source Table name in the Table Occurrence name. This provides a visual and programmatic indicator outside of the Relationship Graph for the underlying table. Given that the Get (LayoutTableName) function returns the Table Occurrence name and not the Source Table Name, and there is no other way to get the Source Table name. In order to programmatically obtain the Source Table Name it needs to be included in the Table Occurrence Name.
- “__” – required
The usage of “__” double underscores is the recommended character. This allows the usage of underscores within Source Table names and Logical Table Occurrence Names while still providing readability and parsing capabilities.

- **DescriptiveName** -
Provides the mechanism to give meaning to the Table Occurrence.
 - **Examples:**
 - FD__Customer__AllCustomers
 - Interface__Invoice__LateInvoices
 - Interface__Invoice__Current_Invoices
 - Synchronization__Version__Host_Stored_Versions
2. **Functional Table Occurrence Group Separation:** Locating a Table Occurrence within the calculation dialog box (while placing a field on a layout or assigning it to a layout) can be difficult. You must select from a drop-down list that lacks any categorization. For example, suppose your solution has 200 Table Occurrences. To perform any of the tasks mentioned above, one would have to navigate through this lengthy list. A TOG convention will help to some degree by listing all the TOGs together in alphabetical order. In the following example we only have 13 Table Occurrences, however, consider a much longer list.

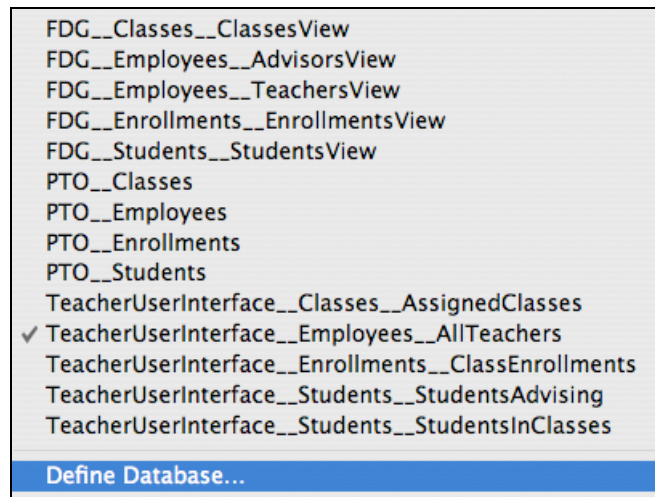


Figure 6

The following shows an example where each FTOG has a header that easily differentiates each TOG. In a long list of TOGs this can be very useful.

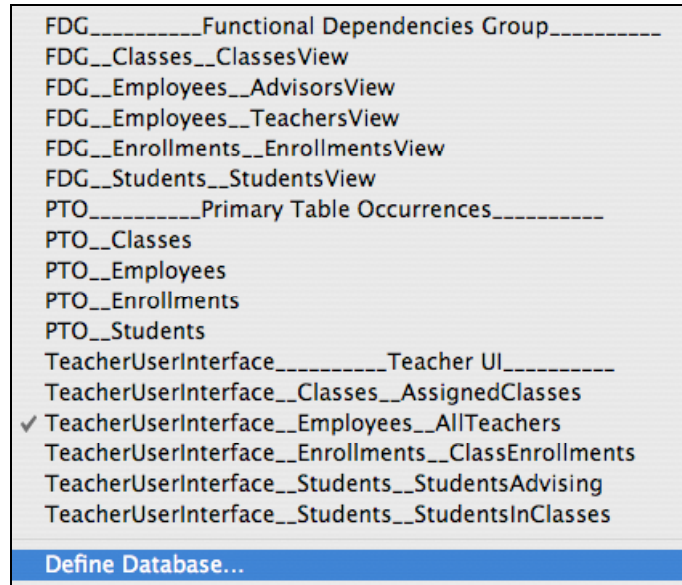


Figure 7

This method requires you to either create a spacing table or use existing Table Occurrences as headers for each functional group. In our example we create a table named 10 hyphens (- - - - -). The name is not important, nor is the creation of a specific table.

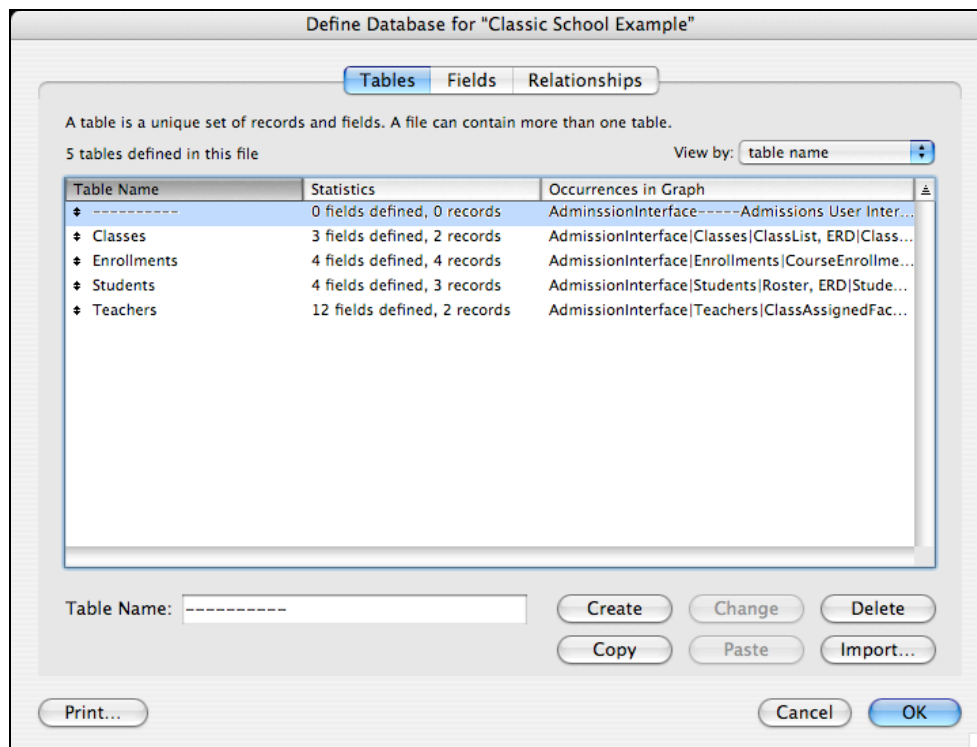


Figure 8

When the Spacing Table is created, its automatically created Table Occurrence will be placed on the Relationships Graph. Add Table Occurrences, using this table as the Source Table, and provide a name that is used to separate each of the functional groups in the drop down list. Our example listed in figure 7 has three of these.

- FDG_____Functional Dependency_____
- PTO_____Primary Table Occurrences_____
- TeacherUserInterface_____Teacher UI_____

In a large list of Table Occurrences this will assist you in finding the Table Occurrence you are looking for by providing headers to each FTOG.

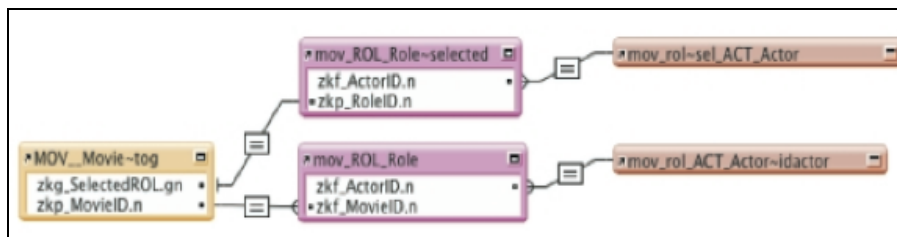
4.3.3 Anchor Buoy / Hierarchical Table Occurrence Grouping (HTOG) Method

This method of Relationship Graph design takes a collection of Table Occurrences (TOs) that are connected by relationship lines, and adds structure and rules.

In the September 2005 issue of FileMaker Advisor, Roger Jacques of Soliant Consulting, explores this method in an article entitled “*Managing the FileMaker Pro 7 Relationship Graph*,” It is impossible to provide the full details of this method in this document. However, I will use his description as the basis for some examples. He provides the following description of the hierarchical/Anchor model.

“The hierarchical nature of the Anchor Method is based on a pattern. Every TOG has an anchor TO and that TO is serviced with related data via any number of threads of buoy TOs. This hierarchical model lends itself to the requirement of only one path between any two TOs. After you start a thread, you might add whatever TOs are required to retrieve related data. Here’s a rule I strictly follow: Don’t join two TOGs with a relationship line and FileMaker will not let you make a circular reference.”

“The anchor TO is always the farthest to the left and the supporting buoy TOs cascade off to the right. You can add TOs as you need them, providing access to related data for the anchor TO. In practice, threads rarely get longer than four or five levels deep.”



“Note that both threads use the same tables, but for different purposes. In this example the lower thread is based on the primary and foreign key relationships for the three entities, and lists all roles and actors for the current move. The upper thread lets users select a portal row for the roles then see details for that role and actor in related fields.”

In this method, layouts may only be based on the “anchor” (leftmost) TO. The “buoy” TOs exists solely to feed data to any layouts or logic based on the anchor.

4.3.3.1 Anchor Buoy / (HTOG) Pros

- Significantly reduces complexity of graph
- Allows the relational structure of the system to be understood, to some degree, outside of the graph.
- Naming is automatic and does not require a lot of thinking or discretion, except for the optional relationship meta-data (Descriptive Name).
- TOGs are mostly layout-based which makes it easy to locate TOs on the Relationships Graph.
- Works very well for large solutions

- Provides functional grouping to Table Occurrence menus outside of the Relationships Graph in a hierarchical order.
- Supports and is unaffected by the “single-path” rule of the Relationship Graph.
- Related tables within a FTOG are fairly limited, and they are grouped into sorted alphabetical order within the FTOG.
- Support a wide range of development methodologies: those under highly scripted “portal-driven” solutions and more open based solutions using FileMaker native controls.
- Is accessible to a wide range of developer, yet robust enough for professionals.

4.3.3.2 Anchor Buoy / (HTOG) Cons

- Significantly increases the number of TOs.
- Does not support bi-directional relation model inherently. The restriction of limiting layouts to only the anchor requires you to create another Anchor Buoy/TOG to express right-to-left data to the layout.

4.3.3.3 Anchor Buoy / HTOG Standards Aware Guidelines

1. Individual words should consistently be separated using the method stated for each section of the syntax.

| Recommended Word Separation Methods (Table Occurrences: Anchor Buoy) | | |
|--|----------------------------------|---|
| Anchors | | |
| AnchorTOGHeaderName | UPPERCASE | |
| SourceTableName | LowerCamelCase | *Should Match Table Name Case |
| Buoys | | |
| AnchorTOGHeaderName | lowerCamelCase | |
| SourceTableNameAbbreviation | UPPERCASE | Abbreviation suggested to minimize length |
| SourceTableName | LowerCamelCase | *Should Match Table Name Case |
| DescriptiveName | lowerCamelCase UpperCamelCase | |

2. Anchor Syntax
 <<AnchorTOGHeaderName>>[]<SourceTableName>
 See [Syntax Legend](#) for description of syntax.

AnchorTOGHeaderName – required; developer defined
 Identifies the Table Occurrence Group (TOG). Every Table Occurrence ‘connected’ to this anchor will have the same Anchor TOG Header Name.

“ ” – required
 Double underscores separates the AnchorTOGHeaderName from the Source Table Name.

SourceTableName –

Indicates the Table Associated with the Table Occurrence.

3. Buoy Syntax

<<AnchorTOGHeaderName>>[_]<<SourceTableNameAbbreviation>>[_]<SourceTableName>[_]DescriptiveName

See [Syntax Legend](#) for description of syntax.

AnchorTOGHeaderName – required; used defined value for anchor

Identifies the Table Occurrence Group (TOG) every Table Occurrence ‘connected’ to the anchor will have the same Anchor TOG Header Name.

“_” – required

Single underscore separates the AnchorTOGHeaderName from the first Source Table Abbreviation Name.

SourceTableNameAbbreviation – required; developer defined

Each SourceTableNameAbbreviation is separated by a single “_” underscore.

“__” – required

Double underscores separates the final SourceTableNameAbbreviation from the Source Table Name

SourceTableName – required; intended value of the actual Source Table Name

Indicates the Table Associated with the Table Occurrence.

“__” – required

Double underscores separates the Source Table Name from the Descriptive Name.

Descriptive Name –

In Buoy Table Occurrences, the Descriptive Name is optional, but can provide a natural description of the TO or perhaps some meta-data indicating what the relationship is based upon. For example, a Descriptive Name of “CompanyID” would indicate the join is based on the CompanyID field.

Examples:

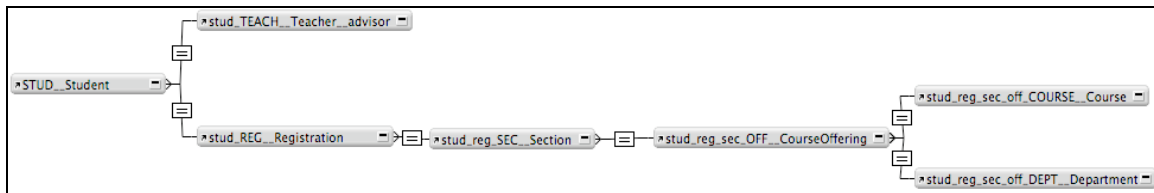


Figure 9

- Anchor STUD__Student – This is an anchor TOG, because it is always the left-most TO and it only lists the Source Table. The name also tells us that it is based on the Source Table “Student”.

- stud_reg_SEC__Section – This buoy TO is part of the “stud” Anchor Group. It is the third TO connected to “reg” which is connected to the anchor “stud”. It is based on the Source Table “Section”. There is no Descriptive Name.
- stud_reg_sec_off__COURSE__CourseIdentification – This buoy TO is part of the “stud” Anchor Group. It is 5 levels deep with a Source Table of “Course”. It is connected to “off” (CourseOffering), “sec” (Section), “reg” (Registration), then the anchor “stud” (Student). It also contains the optional use of Descriptive Name with CourseIdentification.

4.4 Ancillary Considerations

Primary Table Occurrence: Regardless of the Relationship Graph methodology used, when creating calculations you must select the context from which the calculation will evaluate. For those fields where the calculation result is derived without referencing related data, these “internal” calculations should always be based on the same context. The concept of a special Table Occurrence called the Primary Table Occurrence or PTO, can assist in this concern. The PTO always serves as the designated Table Occurrence that will be used when creating internal calculations. Figure 8 demonstrates the usage of the PTO for the source table Employees.

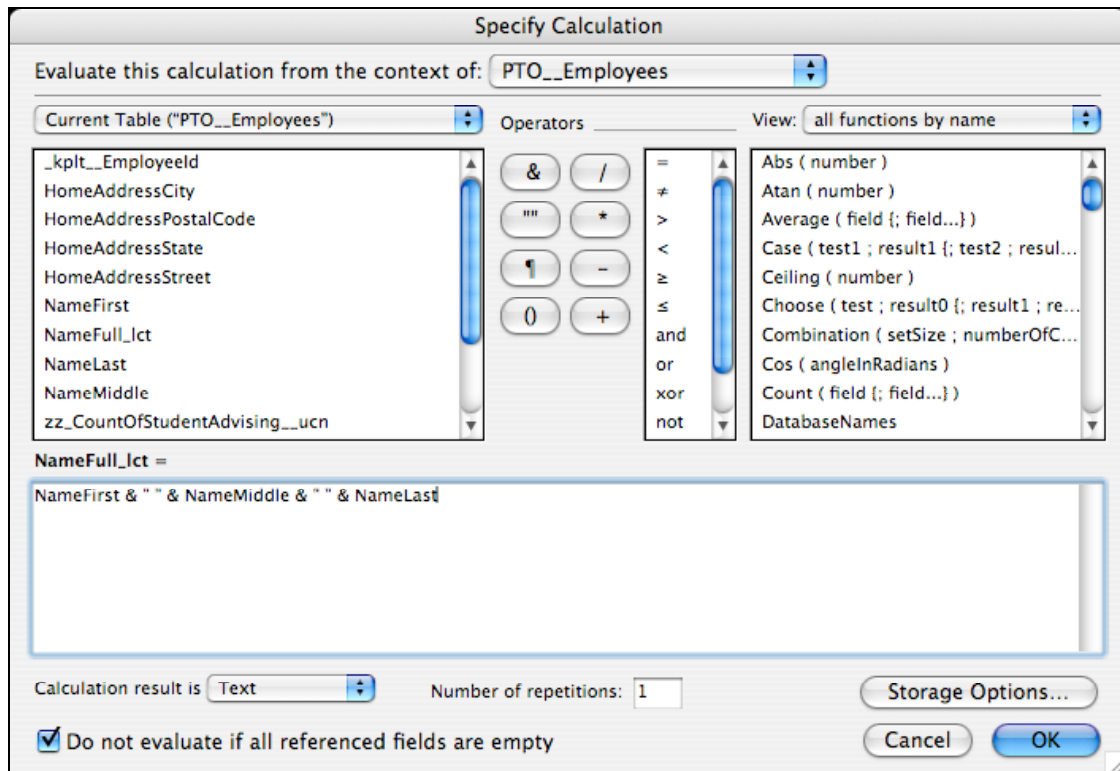


Figure 10

By utilizing a PTO for each internal calculation you will always know that the PTO provides the context in which to evaluate the calculation. As a solution evolves you will not have to remember which series of Table Occurrences are used to evaluate these internal calculations.

Removing other Table Occurrences will have no impact on the calculations using the PTO as the evaluation context.

Syntax:

[PTO][__]<SourceTableName>

Recommended: [PTO__]<<SourceTableName>>

See [Syntax Legend](#) for description of syntax.

- (PTO) PrimaryTableOccurrenceDesignator – required; developer defined; Can be defined by developer. The FDC recommends the use of “PTO.”
- “__” – required
Double underscores serves the purpose of separating the PTO from the Source Table Name.
- SourceTableName –
The name of the associated Table to the Table Occurrence.

5 Layouts

One of the first considerations on how to name layouts is primarily driven by who will use the name. In solutions where the end-user directly selects the layout from the layout pop-up menu in the status area it is important to provide a name that is self-describing and generally not encoded with additional meta-data. Inversely in solutions where the end-user is isolated or prohibited from using the layout pop-up menu, the name can serve the developer.

Other considerations include using a character set that will not encumber external connectivity. For example, XML queries utilize the layout name to establish context. Layout names must be compatible. Again, not all solutions will need to be restricted by this; however, preparation for the evolution of the solution can reduce your development efforts down the road. One should also consider a case convention for consistency and professionalism. The length of a layout name can be up to 100 characters, however, there are some dialog boxes that will not display this length. A common issue is the inability to know what Table Occurrence (TO) is tied to a layout during development. Outside of leaving the context of your development activities, entering layout mode, and entering layout setup, this information is unavailable. Moreover, many solution designs desire the ability to programmatically gain and utilize Source Table assignment to a TO for interface and program logic. The inability to obtain this information from a development or programmatic perspective has lead to the desire to encode or include meta-data about Table Occurrence and Source Table Name within the layout name.

Anytime you utilize notation you make the name less friendly to the end-user. Unfortunately there exists no way to provide both a user-friendly and developer-friendly name to a layout. Therefore, depending on your solution requirements and design you need to consider the trade-offs. And lastly, outside of a structured or encoded name for the layout there exists no way to categorize layouts. For example, layouts are used for interface, reports, 'hidden' work, submitting, and many other specific uses. There is no way to categorize these. This is useful to understand what the purpose of the layout is. This has brought forth the desire to encapsulate some categorization into the layout name.

5.1 Objectives:

- Recommended Character Set
- Recommended Case Convention
- Layout Name Length issues
- Provide method to address the inability to identify the Table Occurrence tied to a layout without entering layout setup
- Provide method to address the challenges programmatically in viewing meta-data about the layouts Source Table Name at runtime
- Provide method to address the inability to programmatically gain meta-data about all the layouts other than the one you are on
- Note the inability to provide a more "user-friendly" name to a more developer useful name
- Note to avoid naming TO and Layout names the same.
- Provide method to address developers' need to understand the "purpose" of the layout
- Provide method to address the inability to organize layouts by function, category, etc.

- Note Only: issues with connectivity technologies (XML, ODBC, JDBC), Including Reserved SQL words

5.2 Problem Definition:

Recommended Character Set: The character set used for layouts should take into consideration the character restrictions imposed by FileMaker itself as well as any technologies that utilize layout names when communicating with FileMaker, such as XML queries.

Recommended Case Convention: This recommendation is not to overcome a specific problem but rather to make a consistent approach on how one will name layouts. The objective is to select a method and be consistent with it.

Layout Name Length: A FileMaker layout can be up to 100 characters in length. This is not a practical limit in many cases. With the release of version 8, most dialog boxes will have no difficulty in displaying the full 100-character name. However the Layout Setup dialog box is not resizable and limits the viewable characters to 42 on OS X and 48 on Windows.

Challenge in identifying the Table Occurrence tied to a layout without entering Layout Setup: There exists no way, outside of a great memory, to know what Table Occurrence a layout is associated with other than entering into Layout Mode and opening the Layout Setup dialog. This lack of information while developing can be cumbersome, requiring dropping in and out of programming activities just to get fundamental layout information. This led to the desire to include notation of the Layouts' Table Occurrence association within the name of the layout.

Challenge to programmatically view meta-data about the layout's Source Table Name at runtime: Developers may want to programmatically gain information about the layout, such as the Table Occurrence Name or Table Name, in order to perform an action a particular way. While the Get (LayoutTableName) will return the Table Occurrence Name, it will not return the actual table name (Source Table name associated with the Table Occurrence). This led to the desire to include the Source Table Name within the layout name.

Challenge to programmatically view meta-data about all the layouts other than the one you are on: Coupled closely to the previous problem is the need to programmatically gain information about all layouts in a file. The Design function LayoutName(FileName) will return all of the layouts in a file. However, there is no function to get Table Occurrences associated with them other than using Get(LayoutTableName) for each individually, and there is no way to get the Source Table Name. This led to the desire to include the Source Table Name and the Table Occurrence Name in the Layout Name.

Challenge to provide a more "user-friendly" name with the developer useful name: Up to and including FileMaker 8, there is no ability to provide both a user-friendly and a developer useful name. As a developer you must make a decision on which method to use for your solution.

Table Occurrence and Layout Names with the same name: By default, FileMaker will generate layout names with the same name as the Table Occurrence the layout is based upon. While this is a time saver in some respects, it can cause a number of problems while attempting to gain information programmatically.

Developers need to understand purpose of layout: When developing a solution the developer in many cases desires to provide some indication of the purpose of the layout in the layout name. For Example, all report layouts might be prefixed with “rep”. Additionally, prefix layouts with “work” might be used indicate the layout is used programmatically to perform some work within a script. Regardless of the categorization, there is a desire to indicate the ‘function’ of the layout both from the perspective of organization and programmatic understanding.

Inability to organize layouts by function, category, and others: Whether you are selecting a layout from the layout tab selection or other dialogs it can be cumbersome to sift through a long list that is only categorized by alphabetic order. The layout list lacks any categorization capabilities. This has lead to the desire to utilize a naming structure that can break down this list into categories or functions to reduce the time to locate layout names.

5.3 Standards Aware Guidelines (Layouts):

1. Should use only the characters
 - Upper & lower case aA – zZ
 - Number: 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Does NOT contain spaces.
3. Does NOT start with numbers.
4. Should be consistent with usage of singular or plural names
5. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Layouts) | |
|---|----------------|
| lowerCamelCase | myLayoutName |
| UpperCamelCase | MyLayoutName |
| Single Underscore (lower Case) | my_layout_name |
| Single Underscore (Title Case) | My_Layout_Name |
| Single Underscore (UPPER CASE) | MY_LAYOUT_NAME |

6. Syntax:
 <<FunctionPrefix>>[_]DescriptiveName[_]<TableOccurrenceName>
 See [Syntax Legend](#) for description of syntax

- Function Prefix – optional; developer defined
 Provides general syntax to allow for developer defined prefixing determined by their specific needs and preferences, while defining a universally understood syntax. If you choose not to use any Functional Prefixing simply omit it and start with the Descriptive Name. There is no need to include the first “_”. The general recommendation is to use a one to five character indicator and select a case convention for the function.

- “__” – required
The usage of a “__” double underscore is the character. This allows the usage of underscores within Descriptive Name and Table Occurrence Name, while still providing readability and parsing capabilities.
- Descriptive Name -
Provides a method to give a ‘somewhat’ user-friendly name to the layout.
- “__” – required
The usage of a “__” double underscore is the character. This allows the usage of underscores within Descriptive Name and Table Occurrence Name, while still providing readability and parsing capabilities.
- Table Occurrence Name – required; uses defined Table Occurrence Name
Provides a visual and programmatic indicator/locator, outside of the Relationship Graph, for the underlying Table Occurrence.

6 Custom Functions

Custom functions provide an unprecedented level of extensibility to the development environment. They provide developers the ability to extend far beyond the default functions built into FileMaker.

FileMaker has a specific syntax and case convention used for all of the built-in functions. For example, the Text Function “MiddleWords” reads MiddleWords (text ; startingWord ; numberOfWords). The function uses UpperCamelCase and each parameter is lowerCamelCase. The only notable exception are the Get functions, which use a different case convention. This is simply due to the nature of a Get function. All Get Functions call “reserved words”. There are no parameters for Get Functions, and thus the syntax is different. The function Get (WindowDesktopHeight) is a good example. Notice that the function name “Get” is always UpperCamelCase and the value it will return is UpperCamelCase as well. For consistency, it makes sense to utilize the case and syntax already established within the product.

Functions can stand along or work in conjunction with any number of other custom functions. There exists no way to categorize or group these related functions. Some suggest that a Custom Function has inputs and outputs and needs no grouping. While others feel that it is important to understand that a function relies on or uses other functions. The FDC has opted to make this distinction by using the terms “Private” and “Public Custom” functions.

When reviewing a calculation it is not always clear that it utilizes a Custom function. It is important to be able to identify a Custom function within a calculation. The calculation engine does not differentiate between built-in and custom functions. This can make it difficult to troubleshoot and understand a calculation. By providing an indicator within the Custom function name it will be easily recognizable within the calculation.

Lastly, one of the key values of Custom Functions is to provide extended functionality that exists only within the function itself. Anyone who uses this function needs to understand what it does and how it works. Therefore, it is important to liberally document for your future reference and for others who may use your function. The FDC recommendation is to provide a base set of guidelines on what and how a Custom function should be documented. In addition, you should refer to the [Calculation section](#) of this document for some guidance on formatting your Custom function calculation.

6.1 Objectives:

- Recommended Character Set
- Recommended Case Convention
- Syntax Separation
- Length Guidelines
- Method to group related functions
- Method to differentiate custom functions within calculations
- Method to differentiate between parent/child functions
- Guidelines on security settings on Private and Public Custom functions
- Method for documenting a custom function including minimum elements and format

6.2 Problem Definition:

Recommended Character Set – Custom functions provide an extension to the default functions in FileMaker Pro/Advanced. For consistency, the recommendation is to follow the character set used within the FileMaker Pro and FileMaker Pro 8 Advanced function list.

Recommended Case Convention – The decision of how to separate words within the Custom function names should be consistent with the built-in functions. All built-in functions follow the same case convention. UpperCamelCase for the name and lowerCamelCase for all parameters is used. Additionally, there needs to be syntax to the Custom function to separate any prefixing needed for organizational needs.

Syntax – The Custom function syntax should incorporate consistency with built-in functions, differentiate prefixes from the name, and support grouping of related functions by name.

Length Recommendations – A Custom function can be up to 100 characters in length. However, one should be aware that lengthy names could be cumbersome to work with in the calculation dialog. The calculation dialog displays 26 characters on OS X and 30 characters on Windows in FileMaker Pro and FileMaker Pro 8 Advanced.

Grouping Related Functions – It is common to see a collection of two or more functions; they work together to provide some functionality. The convention should define a method to allow related functions to be visually recognized, and for developers to understand dependencies among the group.

Differentiating Custom Functions Within Calculations – When interrogating a calculation it is not clear that an element is a built-in or custom function. The convention should define a method to clearly distinguish between the two.

Security on Custom Functions – The convention should provide some general awareness on setting the availability of Public and Private Custom functions.

Documenting Custom Functions – The convention should provide a model for documenting the various elements of a Custom function. These should include the elements, formatting, and location.

6.3 Standards Aware Guidelines (Custom Functions):

6.3.1 Public Custom Functions

A Public Custom function is a custom function intended by its designer to be called directly. These functions might be the "main" functions within a group of Custom functions (that in turn call on Private Custom functions within the group), or they might be individual custom functions that stand alone with no dependencies. In all cases the term refers to functions that are intended to be called directly. For example, assume we have three Custom functions;

- UrlHighlight
- UrlLastChar
- UrlReturnAll
- UrlReturnOne

By utilizing name grouping it is easy to recognize that this set of Custom functions are related. They all are providing some functionality of work to something related to URLs. What is not clear is the structure of the function. Without internal knowledge of the set of functions, it is impossible to know if any one or more of these are dependent upon the other. By classifying each function as Private or Public we can understand which function is designed to be called first.

- UrlHighlight_CFpub
- UrlLastChar_CFpvt
- UrlReturnAll_CFpvt
- UrlReturnOne_CFpvt

This convention tells us that UrlHighlight_CFpub, within the group URL, is designed to be called first and that each of the other functions are subordinate to it.

Syntax:

CustomFunctionName[___][CFpubf]

See [Syntax Legend](#) for description of syntax.

- Suffixed with “_CFpub” (CF uppercase, pub lowercase) – Identifies within any calculation that the function is a custom function of public type.
- Should use only the characters:
 - Upper and lower case aA - zZ
 - Number: 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
- Should NOT contain spaces
- Should NOT start with numbers
- Should NOT contain periods
- Should utilize UpperCamelCase for the function name
- Example: ArrayBuilder_CFpub

6.3.2 Private Custom Functions

The intent of the term "private" is not to signify limited access, but rather to communicate the intent of the function designer: a Private Custom function is one

intended only to be called by another custom function, whether by a Public Custom function or another private custom function. By classifying a custom function as public you are indicating that it is intended be called directly, and that it may have dependencies on other custom functions. Conversely, a Private Custom function is intended always to be called by another custom function, and never to be called directly.

What happens if I do call a "private" function directly?

Nothing! The terms "public" and "private" are merely intended to illustrate a dependency relationship, not to restrict what can be done with a function. In practice, if you intend to call a Private function directly, you may want to rename it as a Public function instead.

Syntax:

CustomFunctionName[___][CFpvtf]

See [Syntax Legend](#) for description of syntax.

- Suffixed with “_CFpvtf” (CF uppercase, pub lowercase) – Identifies within any calculation that the function is a custom function of private type.
- Should use only the characters:
Upper and lower Case aA - zZ
Number: 0,1,2,3,4,5,6,7,8,9
Single and double underscores “_” “__”
- Should NOT contain spaces
- Should NOT start with numbers
- Should NOT contain periods
- Should utilize UpperCamelCase for the function name.
- Example: ArrayBuilderHelper_CFpvt

6.3.3 Custom Function Parameters

In either Private or Public Custom functions, parameters should follow the standard format for all FileMaker functions.

Syntax:

CustomFunctionName[___] [CFpub] or [CFpvt] (parameterOne, parameterTwo)

See [Syntax Legend](#) for description of syntax.

- Should use only the characters:
Upper and lower Case aA - zZ
Number: 0,1,2,3,4,5,6,7,8,9
Single and ouble underscores “_” “__”
- Should NOT contain spaces
- Should NOT start with numbers
- Should NOT contain periods
- Should utilize lowerCamelCase
- Example: MyCustomFunction_CFpub(*parameterOne*, *parameterTwo*)

6.3.4 Custom Function Naming Examples

Figure 11 illustrates a collection of Custom functions utilizing the standards aware naming convention.

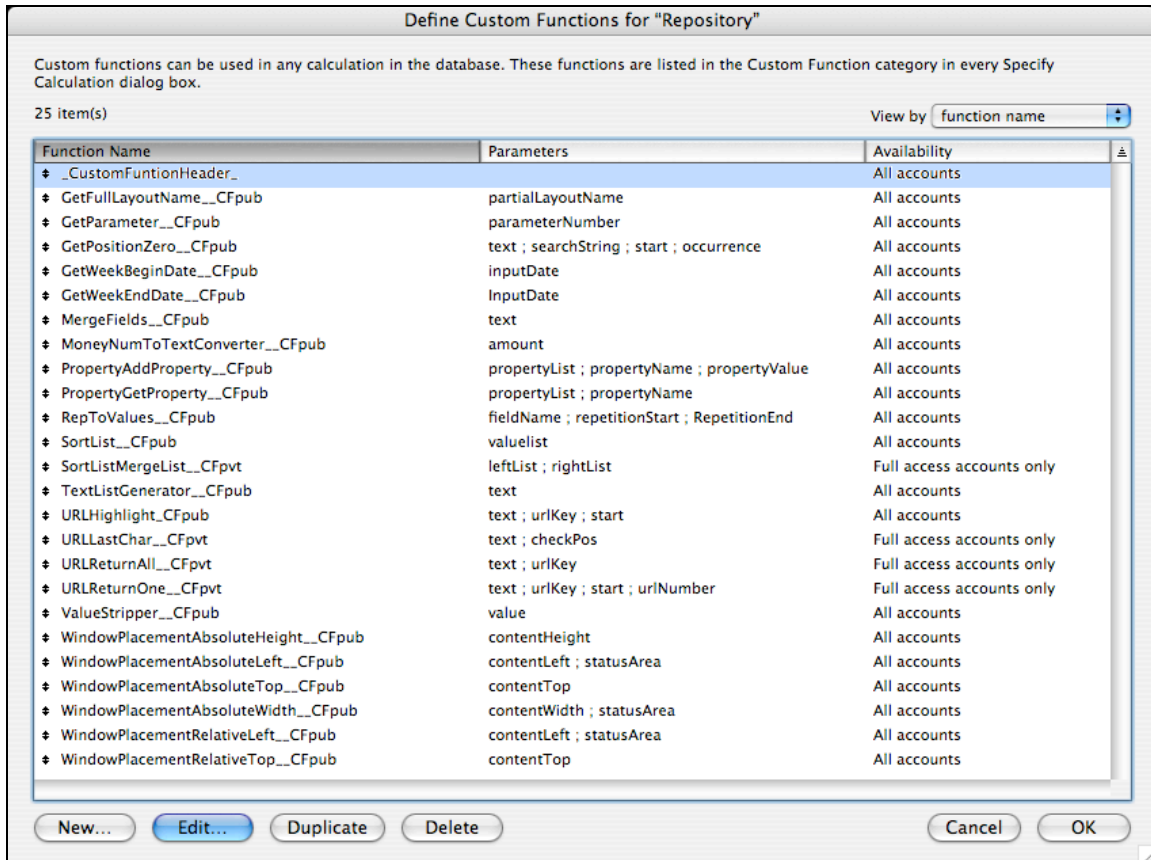


Figure 11

6.3.5 Custom Functions Documentation

Every Custom function should contain a minimum set of documentation. They should be represented on a line by line basis and presented in the following order.

1. Name – Function Name
2. History – (Could include any of the following or others deemed necessary)
 - Creator Name – Name of Function Developer
 - Creator Email – Email of Function Developer
 - Modifier Name – Name of Function Developer who has last modified the function,
 - Modifier email – Email of Function Developer who has last modified the function
 - Date Created – Date in DD-MMM-YYYY the function was created
 - Date Last Modified – Date in DD-MMM-YYYY the function was last modified

3. Purpose – Description of what the function is designed to achieve
4. Parameters – List any parameters used for the function
5. Important Notes – Additional Information
6. Example

/*

Name:

MyCustomFunction_CFpub

History:

Created by John Doe

Creation Date: 01-JAN-2005

Modified Date: 01-JAN-2005

Purpose:

This is an example function that shows the format of documenting.

Parameters:

parameterOne: Example explanation of parameter. This should include any pertinent details.

parameterTwo: Example explanation of parameter. This should include any pertinent details.

Important Notes:

Provide any additional information that would be useful knowledge for future developers and reference.

*/

6.3.6 Custom Function Formatting

See [Calculation Section](#) for Formatting of Custom Function Calculations.

6.4 Ancillary Considerations

Custom unctions can be set to be available in the calculation engine to either “All Accounts” or “Only Accounts Assigned Full Access Privileges”. As a way to enforce privacy you might consider setting your Public Custom functions to “All Accounts” and Private Custom functions to “Only Accounts Assigned Full Access Privileges”. This will make your Public Custom functions available to the calculation engine, while hiding the private ones.

7 Scripts

Managing scripts breaks down into two categories of organization and documentation. First, organizing scripts should really not be considered when it comes to naming. ScriptMaker™ doesn't include organizational tools and many developers have resorted to utilizing their naming of scripts to provide some assistance. While you can manually move scripts up and down a long list there is no real tangible way to categorize a set of scripts. Therefore, position and name are the only options left. This version of the FDC will not attempt to address any nomenclature dealing with organization. However, there are a few areas that apply to documentation and some rules around naming that can be helpful in addressing some hurdles all developers deal with.

7.1 Objectives:

- Recommended Character Set
- Recommended Variable Character Set
- Recommended Case Convention (Variables)
- Script Name Length issues
- Address Inability to recognize a script that requires parameters by viewing the script name
- Inability to provide a more “user-friendly” name or a more developer useful name for those scripts accessible to end-user
- Inability to stop script at last point while using debugger requires an extra script step
- Consistent model for documenting a script
- Challenges with connectivity technologies (XML calling scripts)

7.2 Problem Definition:

Recommended Character Set: There are also some basic restrictions on the characters FileMaker will allow in script names. Additionally when calling scripts from an external call, such as XML, the characters used can present problems. As a developer you need to be aware of the constraints of both FileMaker and any external system your solutions will interact with.

Recommended Case Convention: This recommendation is not to overcome a specific problem but rather to make a consistent approach on how scripts are named. The objective is to select a method and be consistent with it.

Consistent Model for documenting scripts: The convention should provide a model for documenting the various elements of a script. These should include the elements, formatting, and location.

7.3 Standards Aware Guidelines (Scripts):

7.3.1 Script Names

1. Script Names Should use only the characters
 - Upper & lower case aA – zZ
 - Number 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Cannot exceed 100 characters in length

7.3.2 Variables

1. Must start with \$ for local, \$\$ for global
2. Should not contain spaces
3. Individual words should consistently be separated using one of the following methods:

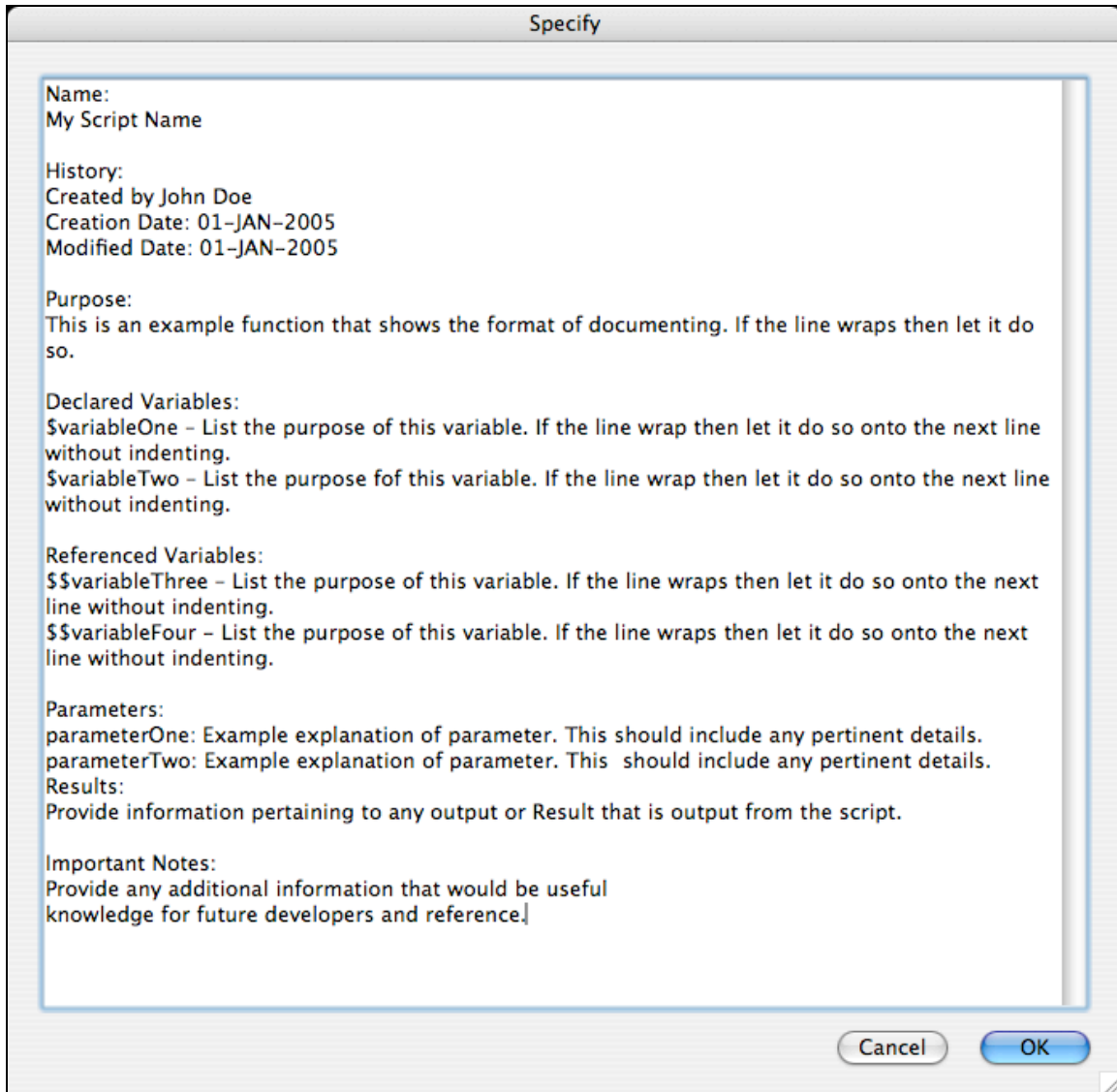
| Recommended Word Separation Methods (Variables) \$,\$\$ | |
|---|--------------------|
| lowerCamelCase | \$myVariableName |
| UpperCamelCase | \$MyVariableName |
| Single Underscore (lower Case) | \$my_variable_name |
| Single Underscore (Title Case) | \$My_Variable_Name |
| Single Underscore (UPPER CASE) | \$MY_VARIABLE_NAME |

7.3.3 Script Documentation

The Comment script step should be used to store the documentation. The comment should be the first step in the script. Comments should include each of the components listed below and be formatted as displayed in the example. If any one comment exceeds the 30K character limit, then an additional comment script step should be added.

1. Name – Script Name
2. History – (Could include any of the following or others deemed necessary)
 - Creator Name – Name of Script Developer
 - Creator Email – Email of Script Developer
 - Modifier Name – Name of Script Developer who has last modified the Script
 - Modifier email – Email of Function Developer who has last modified the function
 - Date Created – Date in DD-MMM-YYYY the script was created
 - Date Last Modified – Date in DD-MMM-YYYY the script was last modified
3. Purpose – Description of what the script is designed to achieve
4. Declared Variables – List any variables the script will be declaring
5. Referenced Variables – List any variables the script will reference, such as previously declared global variables
6. Parameters – List any parameters required for the script
7. Important Notes – Additional Information

Figure 12 shows an example.



The image shows a 'Specify' dialog box with a light gray border and a title bar. Inside, there is a large text area with a light blue border and a vertical scrollbar on the right. The text area contains the following fields and their values:

- Name:**
My Script Name
- History:**
Created by John Doe
Creation Date: 01-JAN-2005
Modified Date: 01-JAN-2005
- Purpose:**
This is an example function that shows the format of documenting. If the line wraps then let it do so.
- Declared Variables:**
\$variableOne – List the purpose of this variable. If the line wrap then let it do so onto the next line without indenting.
\$variableTwo – List the purpose of this variable. If the line wrap then let it do so onto the next line without indenting.
- Referenced Variables:**
\$\$variableThree – List the purpose of this variable. If the line wraps then let it do so onto the next line without indenting.
\$\$variableFour – List the purpose of this variable. If the line wraps then let it do so onto the next line without indenting.
- Parameters:**
parameterOne: Example explanation of parameter. This should include any pertinent details.
parameterTwo: Example explanation of parameter. This should include any pertinent details.
- Results:**
Provide information pertaining to any output or Result that is output from the script.
- Important Notes:**
Provide any additional information that would be useful knowledge for future developers and reference.

At the bottom right of the dialog box, there are two buttons: 'Cancel' and 'OK'.

Figure 12

8 Calculations

The clarity of a calculation formula is increased through formatting and comments. It is up to the developer to provide a meaningful formatting. Good comments should explain the "why" more than the "how." Moreover, good comments should provide an understanding of any code that is not self-descriptive. In some cases better coding techniques can provide clearer understanding. For example, using the Let() function on all but the simplest calculations can help dramatically improve the understanding and readability of the code.

The recommended convention is designed to provide a consistent approach to formatting and commenting calculations. The benefits of formatting and documenting are not obvious for simple calculations; however, as calculations become more complex, it becomes helpful to break them up and document them for future reference, for yourself or the next developer.

8.1 Objectives

- Consistent model for variables used within calculations.
- Consistent model for documenting calculations.
- Recommended comment categories. Provide examples on spacing and formatting to around challenge of color-coding for various components such as: functions, strings, fields, table occurrences, operators and custom functions.

8.2 Problem Definition

Documenting Calculations: Many developers comment to a varying degree. Some take a minimalist approach while others are a bit more verbose. Generally speaking, everyone agrees that some documentation is good, especially with complex calculations. The standards aware recommendation outlines a minimal amount of information as well as a location and format. This can be extended but should contain the minimum components.

Formatting Calculations: Proper formatting of calculations provides better readability and aids in understanding the individual components. Moreover, it makes it much easier for you and your colleagues to make sense of them. Since the calculation engine does not automatically format calculations it's up to the developer to provide this. The FDC has elected to provide general guidance but does not specifically recommend any format.

8.3 Standards Aware Guidelines (Calculations):

8.3.1 Variables

1. Must start with \$ for local, \$\$ for global
2. Should not contain spaces
3. Individual words should consistently be separated using one of the following methods:

| Recommended Word Separation Methods (Variables) \$,\$\$ | |
|---|--------------------|
| lowerCamelCase | \$myVariableName |
| UpperCamelCase | \$MyVariableName |
| Single Underscore (lower Case) | \$my_variable_name |
| Single Underscore (Title Case) | \$My_Variable_Name |
| Single Underscore (UPPER CASE) | \$MY_VARIABLE_NAME |

8.3.2 Calculation Block Header Commenting

Each commented calculation should contain a block header with the following sections:

- Purpose – Provides a general description of what the calculation is designed to do.
- Dependencies – Provides information on what the calculation may depend.
Examples include fields, custom functions and global variables.
- History – Provides a placeholder for any history pertinent to the calculation. This may include the creator, creation date and versioning information.

Example:

```
/*
```

Purpose:

This calculation is used as an example for how one would format a calculation field and properly comment it. The purpose comment will appear in the field list.

Dependencies:

TextField: Text field used by this calculation

NumField: Number field used as a counter

MyCustomFunction__CFpub: Customer Function used in this calculation

History:

Created by John Doe

Date: 1/1/2005

Modified: 2/1/2005 for version x of solution

```
*/
```


8.3.3 In-Line Calculation Commenting

A calculation should be commented in the comments section, not throughout the calculation. One exception is in creating variables with the Let function. Unless the variable name makes its purpose self-evident, the developer should comment the variable. If there is not enough space to comment the variable following its creation, indent the following line and comment after it.

Use indenting liberally to improve readability. Many functions lend themselves to indenting as demonstrated with the Let and Case functions in the following example. This allows parts to be readily separated, such as the calculation body (the Case statement) from the variable creation. (See [Calculation Formatting](#) for more discussion on this topic.)

```
Let(
  [
    Text = TextField1;
    // Put the value of a field into a variable so it's called once
    var1 = expression1;
    // The value of the first expression
    var2 = CustomFunction1 ( Text )
    // Optional commenting line used if the calculation is too long to put
    // the comment after it.
  ];
  Case(
    test1; result1;
    test2; result2;
    defaultResult )
)
```

The FDC recognizes that some calculation fields are extremely simple and their purposes are self-evident. In such calculations, the purpose comment is valuable for documenting the database, but further comments are unnecessary.

8.3.4 Calculation Formatting

Proper formatting of calculations provides better readability and aids in understanding the individual components. Moreover, it makes it much easier for you and your colleagues to make sense of them. Since the calculation engine does not automatically format calculations it's up to the developer to provide this. The FDC provides the following general guidance but does not specifically recommend any format.

8.3.4.1 Formatting Example #1

We can see the following calculation is doing some kind of substitution. However, reading one long string of text makes it rather difficult to know if the Substitute function is the function that starts and ends this calculation. With formatting, you could tell

quickly that the Substitute function begins and ends this calculation, with a carriage return appended at the end.

```
Substitute(DATA::current line; Left(DATA::current line; Position(DATA::current line; "/" ; 1; 1) - 1); Middle(DATA::current line; Position(DATA::current line; "/" ; 1; 1) + 1; Length(DATA::current line) - 1)) & "¶"
```

By glancing at this formatted calculation, you are reminded that the Substitute function requires three parameters: Substitute (text ; searchString ; replaceString).

```
Substitute (  
  
DATA::current line ;  
  
Left (  
DATA::current line;  
Position ( DATA::current line ; "/" ; 1 ; 1 ) - 1  
);  
  
Middle (  
DATA::current line ;  
Position ( DATA::current line ; "/" ; 1 ; 1 ) + 1 ;  
Length ( DATA::current line ) - 1  
)  
  
) & "¶"
```

But when the other functions are split up as well it also makes it easy to understand where those parameters are. See Figure 13 below.

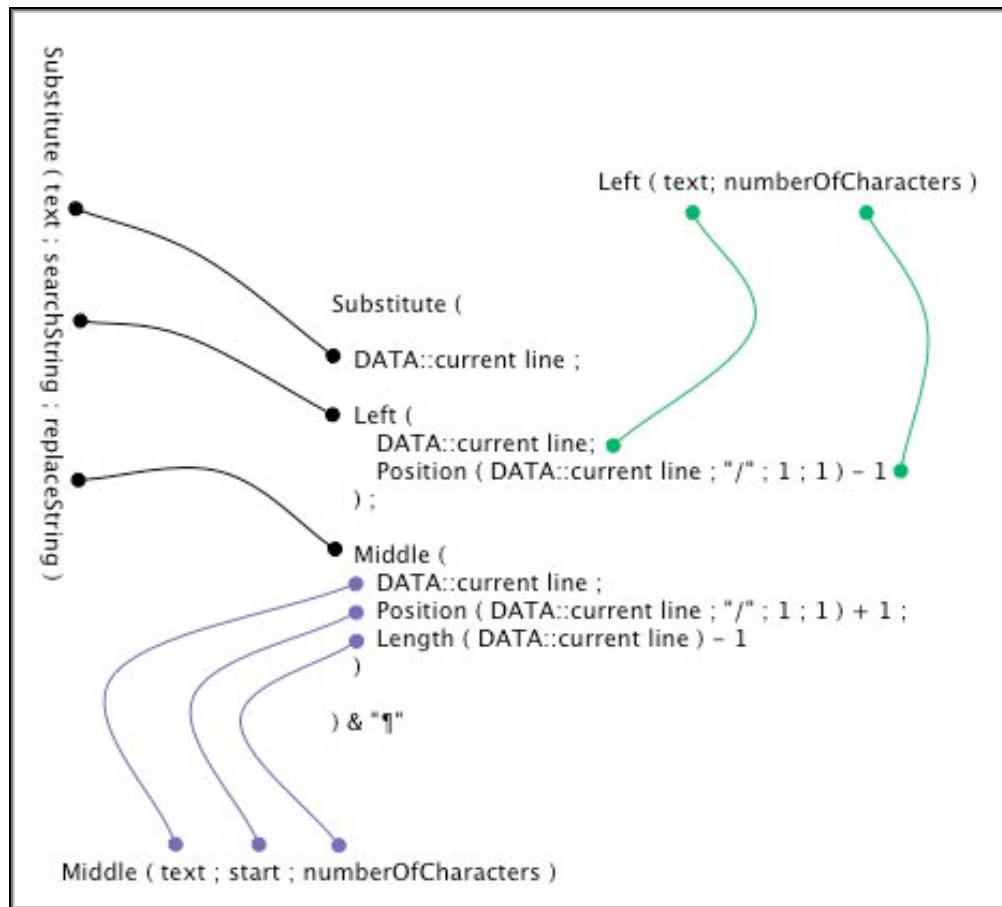


Figure 13

By splitting out each parameter, you can quickly see how much easier it becomes to distinguish the functions.

8.3.4.2 Formatting Example #2

The general principal is to add white space and line breaks to calculations in such a way that it is clear where each function and each function argument begins and ends. If any function call is too long to fit on one line, it is broken up such that:

- The function name and initial parenthesis are followed by a new line.
- Each argument to the function starts a new line and is indented one level deeper than the function name itself.
- The final parenthesis starts a new line, indented at the same level as the function name.

```
If(
  Position( menu_selection; "Alternate Fruit"; 0; 1 );
  Case(
```

```

DayName( Get( CurrentDate ) ) = "Tuesday";
"Oranges";
DayName( Get( CurrentDate ) ) = "Thursday";
"Apples";
/* otherwise */
"Bananas"
);
"Pears"
) & " and " &
If(
  Position( menu_selection; "Alternate Vegetable"; 0; 1 );
  Case(
    DayName( Get( CurrentDate ) ) = "Thursday";
    "Peas";
    /* otherwise */
    "Carrots"
  );
  "Corn"
)

```

One needs to take into account that the FileMaker Pro Calculation dialog does not employ a fixed-width font. Thus, it is necessary to begin the first function argument on a new line, rather than putting both the function call and its first argument on the same line. For example, the following code, while more compact than the above, is less effective when displayed with a variable-width font:

```

If( Position( menu_selection; "Alternate Fruit"; 0; 1 );
  Case( DayName( Get( CurrentDate ) ) = "Tuesday";
    "Oranges";
    DayName( Get( CurrentDate ) ) = "Thursday";
    "Apples";
    /* otherwise */
    "Bananas"
  );
  "Pears"
) & " and " &
If( Position( menu_selection; "Alternate Vegetable"; 0; 1 );
  Case( DayName( Get( CurrentDate ) ) = "Thursday";
    "Peas";
    /* otherwise */
    "Carrots"
  );
  "Corn"
)

```

Use a format where parentheses matching are more obvious. Thus the calculation formatter always starts closing parentheses on a new line when a function is split across multiple lines, rather than returning something like this:

```

If(

```

```

Position( menu_selection; "Alternate Fruit"; 0; 1 );
Case(
    DayName( Get( CurrentDate ) ) = "Tuesday";
    "Oranges";
    DayName( Get( CurrentDate ) ) = "Thursday";
    "Apples";
    /* otherwise */
    "Bananas" );
"Pears" ) & " and " &
If(
    Position( menu_selection; "Alternate Vegetable"; 0; 1 );
    Case(
        DayName( Get( CurrentDate ) ) = "Thursday";
        "Peas";
        /* otherwise */
        "Carrots");
    "Corn" )

```

Additional indentation may be applied on lines, which are the result of "wrapping".

```

Right(
    Middle( 10 ^ 11 + Round( Abs( Amt ); Precision ); 1; 3 ) & "," &
    Middle( 10 ^ 11 + Round( Abs( Amt ); Precision ); 4; 3 );
    Length( Int( Round( Abs( Amt ); Precision ) ) ) +
    If( Precision > 0; 1 + Precision; 0 ) +
    Int(
        ( Length( Int( Round( Abs( Amt ); Precision ) ) ) - 1 ) / 3
    )
)

```

Without indentation for wrapping the result would be:

```

Right(
    Middle( 10 ^ 11 + Round( Abs( Amt ); Precision ); 1; 3 ) & "," &
    Middle( 10 ^ 11 + Round( Abs( Amt ); Precision ); 4; 3 );
    Length( Int( Round( Abs( Amt ); Precision ) ) ) +
    If( Precision > 0; 1 + Precision; 0 ) +
    Int(
        ( Length( Int( Round( Abs( Amt ); Precision ) ) ) - 1 ) / 3
    )
)

```

8.4 Ancillary Considerations

Automated FileMaker Calculation Formatter: Debi Fuchs of Aptworks Consulting has created a “Calculation Formatter” that does most of this work for you. You can try it out at <http://www.aptworx.com/tools>. The calculation formatter does not provide additional indentation on lines, which are the result of wrapping. The developer can augment the formatting of their calculations by adding their own wrapping indentation if desired.

9 Value Lists

The impact of conventions to value lists is somewhat limited. However, some general guidelines can help eliminate common problems, provide extensibility and organize. One should consider using only characters that will not encounter problems with external technologies, such as XML that can reference value list names. You should select and consistently utilize a specific case.

Some developers include meta-data to provide comments or identify a list type. If your development practices lead you in this direction you should consider where you place this meta-data and what form should it take. Placing the meta-data at the end of the name has some benefit. This placement allows you to still use alphabetic sorting for the list names, it allows for type-ahead based on the list name as opposed to the meta-data, and presents the list in an expected format. The other consideration is the form this meta-data should take. The FDC has elected to provide a placeholder that defines some basic notation, but also allows the developer extensibility by allowing this component of the syntax to be defined by the developer for their specific needs. Any extension should be documented in the adherence section of your solution (See Adherence section of this document for more information).

9.1 Objectives:

- Recommended Character Set
- Recommended Case Convention
- Recommended Syntax Separation
- Length Guidelines
- Singular vs. Plural Guidelines
- Address challenge of Value List Comments
- Address challenge of Meta-data (Source/Type) outside of “Define Value List”
- Address inability to organize value lists other than a manual sequential order
- Note: Issues related to updating custom value lists independently of data or schema in a file.
- Calculations link to name rather than internal value list ID

9.2 Problem Definition:

Recommended Character Set : The impact of improper characters in value lists is minimal. However, there are a few considerations. Developers should take care to utilize characters that do not conflict with FileMaker calculations that may be used to interact with value list names. Additionally, developers should be aware of any characters that may present a problem with XML queries. Limiting the character set to the recommendations listed here should help to limit exposure to these issues.

Recommended Case Convention: The decision of how to separate words within value lists has an impact of overall consistency throughout conventions. The recommended convention suggests utilizing only one of the following styles; lowerCamelCase, UpperCamelCase, Underscores, and Uppercase convention for value list names.

Recommended Syntax Separation: Assuming a developer wants to include some meta-data about a value list, even if only to a limited extent, there needs to be a clear understanding for the current and future developers as well as the end-user as to what separates the value list name from the meta-data/notation. The selection of characters for separating meta-data and value list name must be different from that which separates individual words within the value list name.

Value List Name Length: A FileMaker value list can be up to 100 characters in length. This however, is not a practical limit in most cases. With the release of FileMaker 8, all dialog boxes on the Macintosh will have no difficulty in displaying the full name. On the Windows platform there are a few areas where a value list name longer than 30 characters will not display. However, full names are accessible by navigating into the "Define Value List..." dialog. The decision on length is really more a matter of choice rather than a specific limitation.

Singular vs. Plural Guidelines: The decision to utilize singular or collective nouns in place of plural alternatives is noted here only to suggest that as a developer you should make a decision on which you will use and be consistent with it.

9.3 Standards Aware Guidelines (Value Lists):

1. Should use only the characters
 - Upper & lower case aA – zZ
 - Number 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Does NOT contain spaces
3. Is consistent with usage of singular or Plural names
4. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Value Lists) | |
|---|--------------------|
| lowerCamelCase | myValueListName |
| UpperCamelCase | MyValueListName |
| Single Underscore (lower Case) | my_value_list_name |
| Single Underscore (Title Case) | My_Value_List_Name |
| Single Underscore (UPPER CASE) | MY_VALUE_LIST_NAME |

5. Syntax:

ValueListName[___]{suffix}

(See [Syntax Legend](#) at the end of this document for details on syntax.)

Syntax Explained: Value lists syntax is designed to create a consistent approach to value lists that addresses the majority of identified problems. The use of a suffix, instead of a prefix allows for more flexibility of value list names to match the developers style while still addressing some basic consistency. It provides the basic meta-data. It allows for grouping of value list by function.

- ValueListName –
Follow the recommendations outlined above, followed by the “__” double underscores.
- “__” – required
Double underscores acts as separator between value list name and the suffix.
- suffix – optional; developer defined (baseline recommendation provided)
Could be one of the items listed in the following table. This could be extended for any specific needs. Any extension should take the form of a suffix to continue to allow value list items to sort according to name and allow type-ahead functionality based on name rather than meta-data prefixes. Any extension should be documented in the adherence section of your solution. (See [Adherence](#) section of this document for more information). The FDC recommends the following as basic requirement.

Value List Baseline Recommended Type Suffixes

| | |
|---|---|
| c | Custom Value |
| x | Value list from another file |
| d | Value list from field, including all values (for Dynamic) |
| r | Value list from field, including related values (for Related) |

10 Accounts & Security

FileMaker accounts may be setup to authenticate internally or externally. Internal accounts uniquely identify a specific user and are managed within FileMaker entirely. Externally authenticated accounts uniquely identify an external group, which can contain one or more users. External Account membership is managed externally to FileMaker. FileMaker utilizes “privilege sets” to determine the capabilities or rights an authenticated account has to the database. Figure 14 illustrates the relationship between these components.

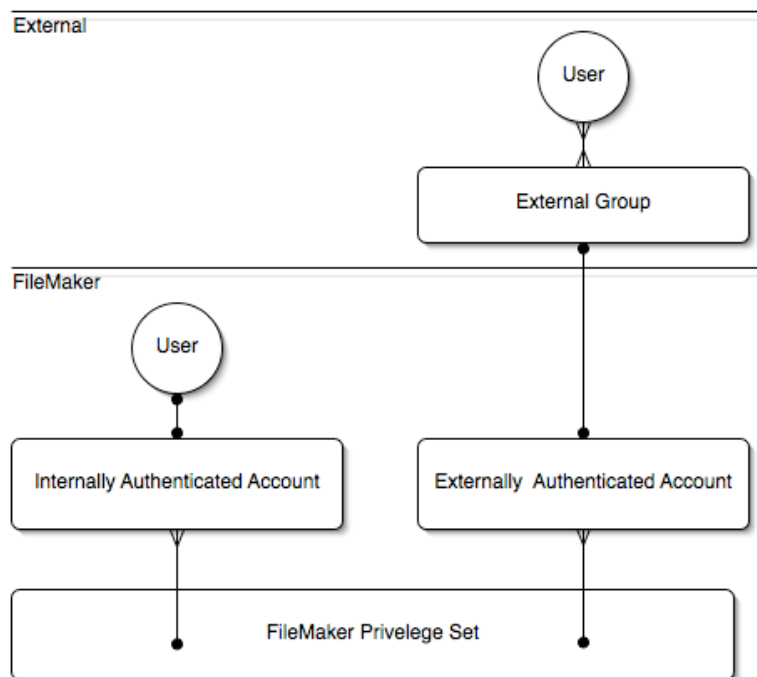


Figure 14

The privilege set defines the capabilities a user or group of users has within the file. Think of this as their role. For example, you may have a developer, administrator, read-only, data entry and various other accounts. The name should represent the role appropriately. This is especially important when administration is ‘outside’ or external to FileMaker.

One of the key benefits of external authentication is the ability to leverage the organization’s security infrastructure. This provides a single place to manage access to all the resources available to any user within the scope of the organization. This also means that IT administrators, not familiar with the specifics of your solution or all the solutions within the environment, will need to assign membership by name. You will need clearly to articulate the file(s)/solution and the role to assign membership with this in mind; some consistency must exist for privilege sets and externally authenticated accounts.

A combination of components needs to be included in the externally authenticated account/group name to identify it. At a minimum it should contain the solution name, to identify the group as being associated with a particular solution. It should also contain the role

for that solution. For Example, suppose we have a solution called “Sales Tracker”. It contains three privilege sets: Operators, NorthAmericanSales, and GlobalSales. Then suppose we have another solution called “Shipping Tracker”, which contains the same privilege sets. Set aside the possibility that each role maps exactly to the same users for each solution, something that will not be the case in many situations. You will need to specify the solution and role a user should be assigned. This requires a naming convention that utilizes both the privilege set and name of the solution for external recognition.

10.1 Objectives:

- Recommended Character Set
- Recommended Case Convention
- Recommended Syntax Separation
- Length Guidelines
- Allow for easy recognition of external group association name to specific solution
- Allow for easy recognition of external group privileges by name to specific solution
- Address duplicate solutions on separate servers when using external authentication

10.2 Problem Definition

Recommended Character Set: There are some basic restrictions on the characters FileMaker and the authenticating OS will allow. To avoid character restrictions the general recommendation is to utilize upper and lower case lower ASCII characters and numbers.

Recommended Case Convention: The choice made here impacts other areas where one applies a convention. The decision is not to overcome a specific problem but rather to introduce a consistent approach on how one will utilize case throughout all other conventions. The objective is to select a method and be consistent with it.

Recommended Syntax Separation: Syntax separation for account naming refers to the way to differentiate the syntax from the names. It is common to see prefixes or suffixes used to indicate a file is part of a group of files. They are also used to hold identification characteristics. We are striving for a consistent way to differentiate this leading/trailing meta-data from the account name.

Length Guidelines: Privilege set and account names are limited to 100 characters. Windows Server 2003 is limited to 64 character group names. Macintosh OS X Server will support 100-character group names.

Solution Recognition: Specific to external authentication. Groups created outside of the development environment must exactly match the names created within the development environment in the FileMaker Pro files. Considering that many organizations will have any number of solutions, it can become cumbersome to the ones relevant to a specific solution. Including the solution name within the group name alleviates this problem.

Privilege (Role) Recognition: Specific to external authentication. Groups created outside of the development environment must exactly match the names created within the development environment. Considering that many organizations will have any number of

solutions, it can become cumbersome to sort through various group names to find the ones relevant to a specific solution. In conjunction with this you need to locate the role for that specific solution as well. Including the role name alleviates this problem.

Duplicate Solutions on Separate Servers: There are situations where the “same” solution is hosted from multiple servers. They are essentially exact copies but serve a different set of users. An extension of the general recommendations should be made to handle this special and somewhat rare situation.

10.3 Standards Aware Guidelines (Security):

10.3.1 Privilege Sets

1. Should use only the characters
 - Upper & lower case aA – zZ
 - Number 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Should not contain spaces.
3. Should consistently use singular or plural names.
4. Can be no longer than 100 characters in length. (When using the recommended externally authenticated accounts guidelines the length, will need to allow for full syntax which can not be more than 64 characters.)
5. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Privilege Sets) | |
|--|-----------------------|
| lowerCamelCase | myPrivilegeSetName |
| UpperCamelCase | MyPrivilegeSetName |
| Single Underscore (lower Case) | my_Privilege_Set_Name |
| Single Underscore (Title Case) | My_Privilege_Set_Name |
| Single Underscore (UPPER CASE) | MY_PRIVILEGE_SET_NAME |

10.3.2 Internally Authenticated Account Names

1. Should use only the characters
 - Upper & lower case aA – zZ
 - Number 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Should not contain spaces.
3. Should consistently use singular or plural names.
4. Can be no longer than 100 characters in length.
5. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Internally Authenticated Accounts) | |
|---|------------------|
| lowerCamelCase | 6. accountName |
| UpperCamelCase | 7. AccountName |
| Single Underscore (lower Case) | 8. account_name |
| Single Underscore (Title Case) | 9. Account_Name |
| Single Underscore (UPPER CASE) | 10. ACCOUNT_NAME |

10.3.3 Externally Authenticated Account Names (Group Names)

1. Should use only the characters

- Upper & lower case aA – zZ
 - Number 0,1,2,3,4,5,6,7,8,9
 - Single and double underscores “_” “__”
2. Should not contain spaces.
 3. Should consistently use singular or plural names.
 4. Should be no longer than 64 characters to work on both the Windows and OS X platforms.
 5. Individual words should consistently be separated using one of the following methods.

| Recommended Word Separation Methods (Internally Authenticated Accounts) | |
|---|--------------|
| lowerCamelCase | accountName |
| UpperCamelCase | AccountName |
| Single Underscore (lower Case) | account_name |
| Single Underscore (Title Case) | Account_Name |
| Single Underscore (UPPER CASE) | ACCOUNT_NAME |

6. Syntax:

<<FileMakerGroupDesignation>>[_]<SolutionName>[_]<PrivilegeSetName>

The syntax is designed to provide a consistent way to name externally authenticated groups. The convention aids in the management of external groups by providing a convention that identifies the group as a “FileMaker Security Group”, a specific solution/file, and the Role or privilege set. With these 3 items, the management of groups is simplified and well organized. We utilize “__” (double underscore to separate each of the sections. These characters are compatible with AD (Active Directory) before and after Windows 2000. They are also supported characters for OD (Open Directory). Before Windows 2000 group names can not contain : ; | = + , * ? < > “ / \ [].

NOTE: There will be those instances where organizational rules may dictate group names. In such cases, utilize the knowledge here to adapt your convention as necessary.

- FileMakerGroupDesignation – required; developer defined
Identifies the group as a FileMaker security group. This will group all FileMaker security groups together within the workgroup manager tool. The recommendation is to use “fm” in lower case. The specific selection is arbitrary but recommended for consistency.
- SolutionName –
Identifies the specific solution/file the group is associated with. This will continue to support the correct sorting by additionally sorting all of the groups for a particular solution together. The name should match your selection made for the primary file or entry point files without the .fpX extension.
- PrivilegeSetName – required; uses privilege set name
Identifies, the role or capabilities the group members have within a solution. Including this in the account/group name provides external management easy identification to the group that needs to be modified.

Special Privilege Sets -

FileMaker has 3 predefined privilege sets: [Full Access], [Data Entry Only], and [Read-Only Access]. These privilege sets are not modifiable, cannot be renamed and cannot be deleted. When using any of these privilege sets, drop the [] brackets from the privilege set name when setting up the external authenticated group account. Windows prohibits these characters in pre-Windows 2000 group names.

[Read Only] = Read_Only, ReadOnly, readOnly, READ_ONLY

Example: fm_MySolution__Read_Only

7. Full Access with Externally Authenticated Accounts: In *FileMaker Security: The Book* by Steven H. Blackwell, he comments on the use of Full Access privilege sets with externally authenticated accounts. He writes; “I strongly recommend that a developer never set an Account with the default [Full Access] Privilege Set to be subject to External Authentication. Authenticate such an Account by the *internal* FileMaker Pro method only. First, all files must have at least one internally authenticated [Full Access] Account. Second, in its initial version FileMaker Server 7 and FileMaker Server 7 Advanced do not support the Global Universal ID (GUID) system. If a hacker or industrial spy obtained a physical copy of a FileMaker Pro file and was able to recreate or to “spoo” the domain structure by guessing Group names, that hacker could gain full and unrestricted access to the file. This is another argument for use of distinctive Group names.” The FDC supports this position.
8. There is one additional special note related to group names as identified in the FileMaker, Inc. technical brief on external server authentication, available at http://www.filemaker.com/downloads/pdf/server_authentication_tb.pdf

“FileMaker Server 7 on Mac OS X Server looks for the group *short name* returned from the Directory Services. That is the official name that identifies the group to the system, not the long (user-friendly) name. Thus, in the definition of accounts in FileMaker Pro 7 for External Server authentication, the defined group name must match the Directory Service Group short name. In many instances the long and short names will be identical; however, in some instances they will not be. Spaces and high ASCII characters might be removed, for example. So we recommend avoiding them altogether. Developers and administrators should check for the short name.”

10.4 Ancillary Considerations:

Host Server Designation: There are situations where the “same” solution is hosted from multiple servers. They are essentially exact copies but serve a different set of users. In these cases you may want to consider an addition to the external group name syntax. The inclusion of the HostServer designation will assist in assigning group membership for the solution on the appropriate server.

Syntax:

<<FileMakerGroupDesignation>>[]<**HostServer**>[]<SolutionName>[]<PrivilegeSetName>

Host Server –Should utilize the DNS, System Name, or Custom Name of the hosting server. Generally these name will be the same, however, this may not be the case in every implementation. Choose the HostServer name that identifies the specific server as recognized by the organization.

Examples:

- Standard Syntax:
 - fm__MySolution__ReadOnly
 - fm__MySolution__General_Users
 - fm__MySolution__salesAssociates
- Using Ancillary Consideration for Host Server:
 - fm__fmserver1__MySolution__General Users
 - fm__fmserver2__MySolution__General Users

II Adherence

This section assumes that you have found value in the content of the FDC and you're ready to implement some standards awareness in your solution based on the recommendations within. Considering that much of the information contained within this document is subject to extension and deviations for a variety of reasons, it makes sense to provide a way to document these differences in a consistent way. Adherence is not so much about strictly following the guidelines as it is about documenting where you differ and/or extend. The objective is to allow the FDC to serve as a baseline, and to reduce the amount of effort you would need to document the how and why of your convention. By allowing the FDC to provide the baseline, you can eliminate a significant portion of this effort. What is needed is for you to indicate where you deviate or extend beyond the FDC. This allows future developers to have reference to both the rationale of the naming with any additional considerations you choose to implement.

Convention Objectives

- Reference the FDC version being utilized
- Reference the FDC version deviations
- Reference any other notable information relative to solution
- Define how information should be stored/accessible
- Define the minimal set of information

Problem Definition

Reference the FDC version being utilized: As the FileMaker product line evolves, the FDC will change to stay current. It follows that a solutions convention will be based on a particular version of the FDC and your specific extensions. Thus, it is important to indicate the version of the FDC.

Reference the FDC version deviations: As with the FDC itself, there is a strong likelihood that your extensions and deviations will change and evolve. Thus, it is important to indicate your deviations with some versioning as well.

Reference any other notable information relative to solution: Some projects/solutions lend themselves to the need for version, creator, and other meta-data elements. However, the concept of release/version applies only within a narrow band project lifecycle, which is during the initial development of a project being built from scratch. These concepts are not too valuable in the "long tail" or "evolutionary" phase of a project. Even if a single developer does most of the initial building, many developers may work on the project over its life. However, for a time period or particular solutions it makes sense to have this meta-data available.

How to store and make accessible adherence information: For consistency this information should be available in a universally understood location.

Minimal set of information: It is generally accepted that every solution should have some documentation. Some systems even contain their own help systems, which contain both user and developer information. Rather than imposing a particular method on building help/documentation systems, the recommendation should simply set the guideline on what the minimum set of information that should be available and how to get to it.

11.1 Standards Aware Convention Guidelines

- I. Every solution should have an “About...” A user or developer with privileges, can open the file should be able to obtain the following items.
 - FDC version utilized
 - URL link to location of FDC on the FileMaker website
 - FDC stored as PDF within a container field that can be exported
 - FDC version deviations
 - URL link to location of version deviation on your website
 - Document stored as PDF within a container field that can be exported that contains all deviations and extension applicable to the solution
 - Documentation meta-data – Any other information, at the discretion of the developer(s), deemed necessary to include. This may be stored in a single field or any number of fields. It should be available from the “About...” interface that is presented. Some examples may include but are not limited to:
 - Company
 - Solution Version
 - Acknowledgements
 - Release Date
 - Developer(s) Name(s)
 - Contact Information

Note:

Runtime developers have legal obligations for “About Layouts”. See the FileMaker 8 Development Guide, Page 82.

Appendix A - SQL Reserved Words

The following table lists all words reserved in the SQL standard

Source: <http://developer.mimer.se/validator/sql-reserved-words.tml>

| SQL-92 | SQL-99 | SQL-2003 |
|---------------|---------------|---------------|
| ABSOLUTE | ABSOLUTE | |
| ACTION | ACTION | |
| ADD | ADD | ADD |
| | AFTER | |
| ALL | ALL | ALL |
| ALLOCATE | ALLOCATE | ALLOCATE |
| ALTER | ALTER | ALTER |
| AND | AND | AND |
| ANY | ANY | ANY |
| ARE | ARE | ARE |
| | ARRAY | ARRAY |
| AS | AS | AS |
| ASC | ASC | |
| | ASENSITIVE | ASENSITIVE |
| ASSERTION | ASSERTION | |
| | ASYMMETRIC | ASYMMETRIC |
| AT | AT | AT |
| | ATOMIC | ATOMIC |
| AUTHORIZATION | AUTHORIZATION | AUTHORIZATION |
| AVG | | |
| | BEFORE | |
| BEGIN | BEGIN | BEGIN |
| BETWEEN | BETWEEN | BETWEEN |
| | | BIGINT |
| | BINARY | BINARY |
| BIT | BIT | |
| BIT_LENGTH | | |
| | BLOB | BLOB |
| | BOOLEAN | BOOLEAN |
| BOTH | BOTH | BOTH |
| | BREADTH | |
| BY | BY | BY |
| CALL | CALL | CALL |
| | CALLED | CALLED |
| CASCADE | CASCADE | |
| CASCADED | CASCADED | CASCADED |
| CASE | CASE | CASE |
| CAST | CAST | CAST |
| CATALOG | CATALOG | |
| CHAR | CHAR | CHAR |
| CHAR_LENGTH | | |

| SQL-92 | SQL-99 | SQL-2003 |
|-------------------|----------------------------------|----------------------------------|
| CHARACTER | CHARACTER | CHARACTER |
| CHARACTER_LENGTH | | |
| CHECK | CHECK | CHECK |
| | CLOB | CLOB |
| CLOSE | CLOSE | CLOSE |
| COALESCE | | |
| COLLATE | COLLATE | COLLATE |
| COLLATION | COLLATION | |
| COLUMN | COLUMN | COLUMN |
| COMMIT | COMMIT | COMMIT |
| CONDITION | CONDITION | CONDITION |
| CONNECT | CONNECT | CONNECT |
| CONNECTION | CONNECTION | |
| CONSTRAINT | CONSTRAINT | CONSTRAINT |
| CONSTRAINTS | CONSTRAINTS | |
| | CONSTRUCTOR | |
| CONTAINS | | |
| CONTINUE | CONTINUE | CONTINUE |
| CONVERT | | |
| CORRESPONDING | CORRESPONDING | CORRESPONDING |
| COUNT | | |
| CREATE | CREATE | CREATE |
| CROSS | CROSS | CROSS |
| | CUBE | CUBE |
| CURRENT | CURRENT | CURRENT |
| CURRENT_DATE | CURRENT_DATE | CURRENT_DATE |
| | CURRENT_DEFAULT_TRANSFORM_GROUP | CURRENT_DEFAULT_TRANSFORM_GROUP |
| CURRENT_PATH | CURRENT_PATH | CURRENT_PATH |
| | CURRENT_ROLE | CURRENT_ROLE |
| CURRENT_TIME | CURRENT_TIME | CURRENT_TIME |
| CURRENT_TIMESTAMP | CURRENT_TIMESTAMP | CURRENT_TIMESTAMP |
| | CURRENT_TRANSFORM_GROUP_FOR_TYPE | CURRENT_TRANSFORM_GROUP_FOR_TYPE |
| CURRENT_USER | CURRENT_USER | CURRENT_USER |
| CURSOR | CURSOR | CURSOR |
| | CYCLE | CYCLE |
| | DATA | |
| DATE | DATE | DATE |
| DAY | DAY | DAY |
| DEALLOCATE | DEALLOCATE | DEALLOCATE |
| DEC | DEC | DEC |
| DECIMAL | DECIMAL | DECIMAL |
| DECLARE | DECLARE | DECLARE |
| DEFAULT | DEFAULT | DEFAULT |
| DEFERRABLE | DEFERRABLE | |
| DEFERRED | DEFERRED | |
| DELETE | DELETE | DELETE |

| SQL-92 | SQL-99 | SQL-2003 |
|---------------|---------------|---------------|
| | DEPTH | |
| | DEREF | DEREF |
| DESC | DESC | |
| DESCRIBE | DESCRIBE | DESCRIBE |
| DESCRIPTOR | DESCRIPTOR | |
| DETERMINISTIC | DETERMINISTIC | DETERMINISTIC |
| DIAGNOSTICS | DIAGNOSTICS | |
| DISCONNECT | DISCONNECT | DISCONNECT |
| DISTINCT | DISTINCT | DISTINCT |
| DO | DO | DO |
| DOMAIN | DOMAIN | |
| DOUBLE | DOUBLE | DOUBLE |
| DROP | DROP | DROP |
| | DYNAMIC | DYNAMIC |
| | EACH | EACH |
| | | ELEMENT |
| ELSE | ELSE | ELSE |
| ELSEIF | ELSEIF | ELSEIF |
| END | END | END |
| | EQUALS | |
| ESCAPE | ESCAPE | ESCAPE |
| EXCEPT | EXCEPT | EXCEPT |
| EXCEPTION | EXCEPTION | |
| EXEC | EXEC | EXEC |
| EXECUTE | EXECUTE | EXECUTE |
| EXISTS | EXISTS | EXISTS |
| EXIT | EXIT | EXIT |
| EXTERNAL | EXTERNAL | EXTERNAL |
| EXTRACT | | |
| FALSE | FALSE | FALSE |
| FETCH | FETCH | FETCH |
| | FILTER | FILTER |
| FIRST | FIRST | |
| FLOAT | FLOAT | FLOAT |
| FOR | FOR | FOR |
| FOREIGN | FOREIGN | FOREIGN |
| FOUND | FOUND | |
| | FREE | FREE |
| FROM | FROM | FROM |
| FULL | FULL | FULL |
| FUNCTION | FUNCTION | FUNCTION |
| | GENERAL | |
| GET | GET | GET |
| GLOBAL | GLOBAL | GLOBAL |
| GO | GO | |
| GOTO | GOTO | |

| SQL-92 | SQL-99 | SQL-2003 |
|---------------|----------------|-----------------|
| GRANT | GRANT | GRANT |
| GROUP | GROUP | GROUP |
| | GROUPING | GROUPING |
| HANDLER | HANDLER | HANDLER |
| HAVING | HAVING | HAVING |
| | HOLD | HOLD |
| HOUR | HOUR | HOUR |
| IDENTITY | IDENTITY | IDENTITY |
| IF | IF | IF |
| IMMEDIATE | IMMEDIATE | IMMEDIATE |
| IN | IN | IN |
| INDICATOR | INDICATOR | INDICATOR |
| INITIALLY | INITIALLY | |
| INNER | INNER | INNER |
| INOUT | INOUT | INOUT |
| INPUT | INPUT | INPUT |
| INSENSITIVE | INSENSITIVE | INSENSITIVE |
| INSERT | INSERT | INSERT |
| INT | INT | INT |
| INTEGER | INTEGER | INTEGER |
| INTERSECT | INTERSECT | INTERSECT |
| INTERVAL | INTERVAL | INTERVAL |
| INTO | INTO | INTO |
| IS | IS | IS |
| ISOLATION | ISOLATION | |
| | ITERATE | ITERATE |
| JOIN | JOIN | JOIN |
| KEY | KEY | |
| LANGUAGE | LANGUAGE | LANGUAGE |
| | LARGE | LARGE |
| LAST | LAST | |
| | LATERAL | LATERAL |
| LEADING | LEADING | LEADING |
| LEAVE | LEAVE | LEAVE |
| LEFT | LEFT | LEFT |
| LEVEL | LEVEL | |
| LIKE | LIKE | LIKE |
| LOCAL | LOCAL | LOCAL |
| | LOCALTIME | LOCALTIME |
| | LOCALTIMESTAMP | LOCALTIMESTAMP |
| | LOCATOR | |
| LOOP | LOOP | LOOP |
| LOWER | | |
| | MAP | |
| MATCH | MATCH | MATCH |
| MAX | | |

| SQL-92 | SQL-99 | SQL-2003 |
|--------------|------------|-----------|
| | | MEMBER |
| | | MERGE |
| | METHOD | METHOD |
| MIN | | |
| MINUTE | MINUTE | MINUTE |
| | MODIFIES | MODIFIES |
| MODULE | MODULE | MODULE |
| MONTH | MONTH | MONTH |
| | | MULTISET |
| NAMES | NAMES | |
| NATIONAL | NATIONAL | NATIONAL |
| NATURAL | NATURAL | NATURAL |
| NCHAR | NCHAR | NCHAR |
| | NCLOB | NCLOB |
| | NEW | NEW |
| NEXT | NEXT | |
| NO | NO | NO |
| | NONE | NONE |
| NOT | NOT | NOT |
| NULL | NULL | NULL |
| NULLIF | | |
| NUMERIC | NUMERIC | NUMERIC |
| | OBJECT | |
| OCTET_LENGTH | | |
| OF | OF | OF |
| | OLD | OLD |
| ON | ON | ON |
| ONLY | ONLY | ONLY |
| OPEN | OPEN | OPEN |
| OPTION | OPTION | |
| OR | OR | OR |
| ORDER | ORDER | ORDER |
| | ORDINALITY | |
| OUT | OUT | OUT |
| OUTER | OUTER | OUTER |
| OUTPUT | OUTPUT | OUTPUT |
| | OVER | OVER |
| OVERLAPS | OVERLAPS | OVERLAPS |
| PAD | PAD | |
| PARAMETER | PARAMETER | PARAMETER |
| PARTIAL | PARTIAL | |
| | PARTITION | PARTITION |
| PATH | PATH | |
| POSITION | | |
| PRECISION | PRECISION | PRECISION |
| PREPARE | PREPARE | PREPARE |

| SQL-92 | SQL-99 | SQL-2003 |
|---------------|---------------|-----------------|
| PRESERVE | PRESERVE | |
| PRIMARY | PRIMARY | PRIMARY |
| PRIOR | PRIOR | |
| PRIVILEGES | PRIVILEGES | |
| PROCEDURE | PROCEDURE | PROCEDURE |
| PUBLIC | PUBLIC | |
| | RANGE | RANGE |
| READ | READ | |
| | READS | READS |
| REAL | REAL | REAL |
| | RECURSIVE | RECURSIVE |
| | REF | REF |
| REFERENCES | REFERENCES | REFERENCES |
| | REFERENCING | REFERENCING |
| RELATIVE | RELATIVE | |
| | RELEASE | RELEASE |
| REPEAT | REPEAT | REPEAT |
| RESIGNAL | RESIGNAL | RESIGNAL |
| RESTRICT | RESTRICT | |
| | RESULT | RESULT |
| RETURN | RETURN | RETURN |
| RETURNS | RETURNS | RETURNS |
| REVOKE | REVOKE | REVOKE |
| RIGHT | RIGHT | RIGHT |
| | ROLE | |
| ROLLBACK | ROLLBACK | ROLLBACK |
| | ROLLUP | ROLLUP |
| ROUTINE | ROUTINE | |
| | ROW | ROW |
| ROWS | ROWS | ROWS |
| | SAVEPOINT | SAVEPOINT |
| SCHEMA | SCHEMA | |
| | SCOPE | SCOPE |
| SCROLL | SCROLL | SCROLL |
| | SEARCH | SEARCH |
| SECOND | SECOND | SECOND |
| SECTION | SECTION | |
| SELECT | SELECT | SELECT |
| | SENSITIVE | SENSITIVE |
| SESSION | SESSION | |
| SESSION_USER | SESSION_USER | SESSION_USER |
| SET | SET | SET |
| | SETS | |
| SIGNAL | SIGNAL | SIGNAL |
| | SIMILAR | SIMILAR |
| SIZE | SIZE | |

| SQL-92 | SQL-99 | SQL-2003 |
|-----------------|-----------------|-----------------|
| SMALLINT | SMALLINT | SMALLINT |
| SOME | SOME | SOME |
| SPACE | SPACE | |
| SPECIFIC | SPECIFIC | SPECIFIC |
| | SPECIFICTYPE | SPECIFICTYPE |
| SQL | SQL | SQL |
| SQLCODE | | |
| SQLERROR | | |
| SQLException | SQLException | SQLException |
| SQLSTATE | SQLSTATE | SQLSTATE |
| SQLWARNING | SQLWARNING | SQLWARNING |
| | START | START |
| | STATE | |
| | STATIC | STATIC |
| | | SUBMULTISET |
| SUBSTRING | | |
| SUM | | |
| | SYMMETRIC | SYMMETRIC |
| | SYSTEM | SYSTEM |
| SYSTEM_USER | SYSTEM_USER | SYSTEM_USER |
| TABLE | TABLE | TABLE |
| | | TABLESAMPLE |
| TEMPORARY | TEMPORARY | |
| THEN | THEN | THEN |
| TIME | TIME | TIME |
| TIMESTAMP | TIMESTAMP | TIMESTAMP |
| TIMEZONE_HOUR | TIMEZONE_HOUR | TIMEZONE_HOUR |
| TIMEZONE_MINUTE | TIMEZONE_MINUTE | TIMEZONE_MINUTE |
| TO | TO | TO |
| TRAILING | TRAILING | TRAILING |
| TRANSACTION | TRANSACTION | |
| TRANSLATE | | |
| TRANSLATION | TRANSLATION | TRANSLATION |
| | TREAT | TREAT |
| | TRIGGER | TRIGGER |
| TRIM | | |
| TRUE | TRUE | TRUE |
| | UNDER | |
| UNDO | UNDO | UNDO |
| UNION | UNION | UNION |
| UNIQUE | UNIQUE | UNIQUE |
| UNKNOWN | UNKNOWN | UNKNOWN |
| | UNNEST | UNNEST |
| UNTIL | UNTIL | UNTIL |
| UPDATE | UPDATE | UPDATE |
| UPPER | | |

| SQL-92 | SQL-99 | SQL-2003 |
|---------------|---------------|-----------------|
| USAGE | USAGE | |
| USER | USER | USER |
| USING | USING | USING |
| VALUE | VALUE | VALUE |
| VALUES | VALUES | VALUES |
| VARCHAR | VARCHAR | VARCHAR |
| VARYING | VARYING | VARYING |
| VIEW | VIEW | |
| WHEN | WHEN | WHEN |
| WHENEVER | WHENEVER | WHENEVER |
| WHERE | WHERE | WHERE |
| WHILE | WHILE | WHILE |
| | WINDOW | WINDOW |
| WITH | WITH | WITH |
| | WITHIN | WITHIN |
| | WITHOUT | WITHOUT |
| WORK | WORK | |
| WRITE | WRITE | |
| YEAR | YEAR | YEAR |
| ZONE | ZONE | |
| | | |

Appendix B - Character Usage Chart

[X] Not Allowed or Warning on Attempt [NR] Not Recommended [C] Convention Uses [R] Reserved Character

| | Character | File Names | Tables Names | TO Names | Field Names | Layout Name | Custom Function Name | Value List Names | Scripts Names | Account Names | Variables |
|----|----------------------|------------|--------------|----------|-------------|-------------|----------------------|------------------|---------------|---------------|-----------|
| . | Period | X | X | X | X | NR | NR | NR | NR | NR | NR |
| + | Plus or Addition | NR | X | X | X | NR | X | NR | NR | NR | NR |
| * | Star | NR | X | X | X | NR | X | NR | NR | NR | NR |
| ^ | Carrot | NR | X | X | X | NR | X | NR | NR | NR | NR |
| = | Equals | NR | X | X | X | NR | X | NR | NR | NR | NR |
| > | Greater Than | NR | X | X | X | NR | X | NR | NR | NR | NR |
| (| Left Parentheses | NR | X | X | X | NR | X | NR | NR | NR | NR |
| " | Double Quote | NR | X | X | X | NR | X | NR | NR | NR | NR |
| : | Colon | NR | NR | NR | X | NR | NR | NR | NR | NR | NR |
| , | Comma | NR | X | X | X | NR | X | NR | NR | NR | NR |
| - | Minus or Subtraction | NR | X | X | X | NR | X | NR | NR | NR | NR |
| / | Forward Slash | NR | X | X | X | NR | X | NR | NR | NR | NR |
| & | Ampersand | NR | X | X | X | NR | X | NR | NR | NR | NR |
| ≠ | Not Equals | NR | X | X | X | NR | X | NR | NR | NR | NR |
| < | Less Than | NR | X | X | X | NR | X | NR | NR | NR | NR |
|) | Right Parentheses | NR | X | X | X | NR | X | NR | NR | NR | NR |
| ; | Semicolon | NR | X | X | X | NR | X | NR | NR | NR | NR |
| :: | Relational Indicator | NR | NR | X | X | NR | X | NR | NR | NR | NR |
| { | Left Curly Brace | NR | NR | X | NR | NR | NR | NR | NR | NR | NR |

| | | | | | | | | | | | |
|------|--------------------|----|----|----|----|----|----|----|----|----|----|
| } | Right Curly Brace | NR | X | X | NR | NR | NR | NR | NR | NR | NR |
| ? | Question | NR | NR | NR | NR | NR | NR | NR | NR | NR | NR |
| ~ | Tilde | NR | NR | NR | NR | NR | NR | NR | NR | NR | NR |
| ' | Apostrophe | NR | NR | NR | NR | NR | NR | NR | NR | NR | NR |
| ! | Exclamation | NR | NR | NR | NR | NR | NR | NR | NR | NR | NR |
| % | Percentage | NR | NR | X | NR | NR | NR | NR | NR | NR | NR |
| | Pipe | NR | NR | R | NR | NR | NR | NR | NR | NR | NR |
| \$ | Dollar | NR | X | X | X | NR | NR | NR | NR | NR | R |
| \$\$ | Double Dollar | NR | X | X | X | NR | NR | NR | NR | NR | R |
| [| Left Bracket | NR | X | X | X | NR | X | NR | NR | NR | NR |
|] | Right Bracket | NR | X | X | X | NR | X | NR | NR | NR | NR |
| — | Double underscores | C | C | C | C | C | C | C | C | C | NR |

Appendix C – Terminology & Definitions

Camel Case: The practice of writing compound words or phrases where the words are joined without spaces, and each word is capitalized within the compound. The name comes from the uppercase "bumps" in the middle of the compound word, suggesting the humps of a camel.

```
camelCaseLooksLikeThis
lowerCamelCaseLooksTheSame
UpperCamelCaseLooksLikeThis
```

There are two common varieties of CamelCase, distinguished by their handling of the initial letter. The variety in which the first letter is capitalized is commonly called UpperCamelCase, Pascal case, or BiCapitalized. The variety in which the first letter is left as lowercase is commonly called lowerCamelCase or sometimes simply camelCase. For clarity, this document will use the terms UpperCamelCase and lowerCamelCase, respectively.

Source: <http://en.wikipedia.org/wiki/CamelCase>

Public Custom Function: A Public custom function is the first called function, or parent, within a custom function group. There are many instances where a 'collection' or 'group' of custom functions are used to provide a full set of functionality. In other cases a custom function may exist by itself. In either case the custom function that is called directly is considered to be a Public custom function.

Private Custom Function: A private custom function is never called directly in a calculation. It is only called from another Public custom function. Private functions are subordinate to Public functions. It is very possible to have a custom function that is always private. For example, one might create a custom function that is designed to be a 'helper' function to multiple other functions. In other words, it will only ever be used in conjunction with another custom function.

Primary Table Occurrence: PTO is a special table occurrence. The PTO serves as the designated table occurrence that will be used when creating calculations that are internally referenced. Those calculations are derived from data contained within the context of the table itself and never data from another table.

Spacing Table: A technique used to create 'labels' or 'separator' tables within the Define Database dialog. This technique uses tables that contain no fields for the purpose of grouping and categorizing tables and table occurrences.

Table: A collection of data pertaining to a subject, such as customers or stock prices. A database file contains one or more tables, which consist of fields and records. When you create a new table, a visual representation, or table occurrence, of the table appears in the Relationship graph. You can specify multiple occurrences (with unique names) of the same table in order to work with complex relationships in the graph.

Table Occurrence: A table occurrence refers to an instance of a table on the Relationships Graph. Keep in mind that all interactions throughout the development environment will interact with table occurrences. This is the one and only way to 'address' a table and its contents.

Source Table: A table occurrence is associated with a table. The source table refers to the table the table occurrence is associated with. For example, a table occurrence named “AdmissionInterface|Classes|ClassList” has a source table named “classes”. Source table will also be referred to as the “source table name”.

Logical Solution Identifier: LSI is a prefix used for each secondary file and optional suffix for each primary file in a solution.

FileMaker Group Designator: is a prefix used to identify externally authenticated groups.

Appendix D - Syntax Legend

The FDC document uses a specific style to indicate the various components of any syntax used within the various sections. The following table explains the various notations.

| FDC Syntax Legend | |
|-------------------|--|
| | No Enclosing Characters indicates optional and natural/free form naming |
| [] | Enclosed components are intended to be used as displayed |
| { } | Enclosed components are optional and definable by developer |
| () | Enclosed components are optional but have a defined or inferred value if used |
| < > | Enclosed components are intended to be supplied but have a defined or inferred value |
| << >> | Enclosed components are intended to be supplied and definable by developer |