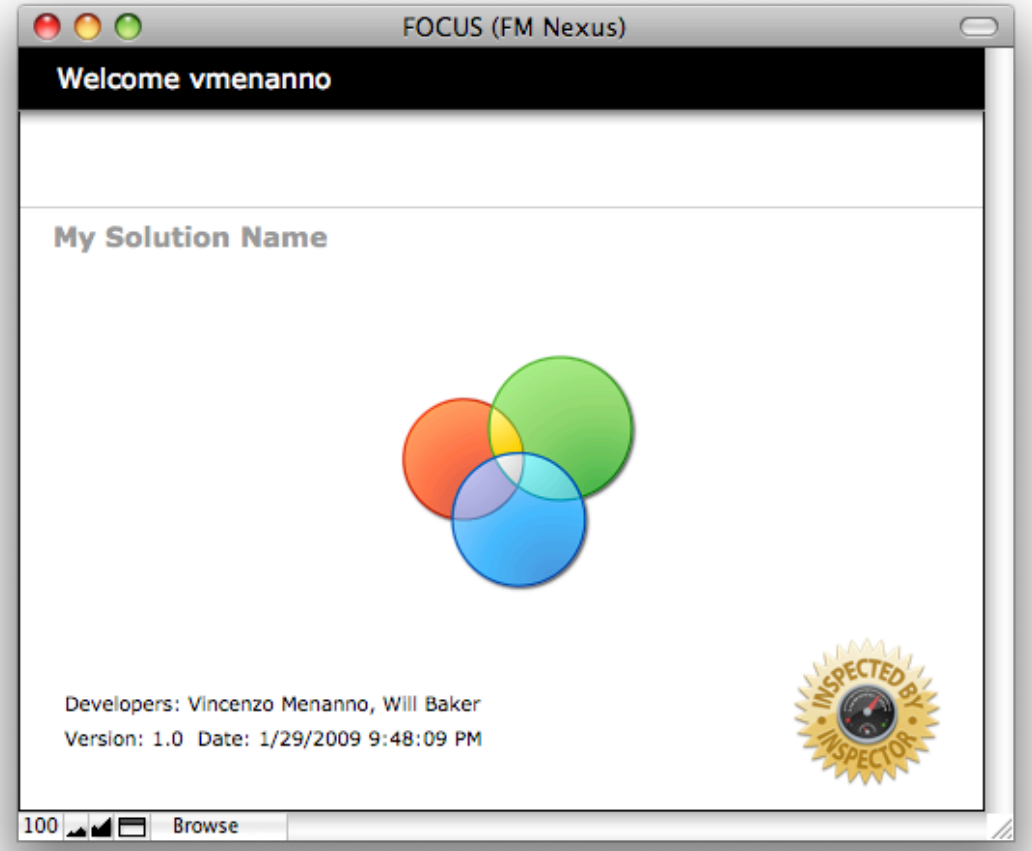
FOCUS Framework



Beezwax Datatools Inc.
888 835-4483
info@beezwax.net

Vincenzo Menanno
510 384-4483
vince@beezwax.net

Will M. Baker
510-316-7601
will_b@beezwax.net

Erin McAnallen
415-531-7835
erin_m@beezwax.net

What is FOCUS?

As much as FOCUS is a set of guidelines and a way to develop and design FileMaker solutions, it is also a starter file that will help get a project started. This starter file is chock full of useful additions that have come from our own experiences as developers as well as the great contributions of other talented developers.

The FOCUS framework is a continually evolving and changing thing. We even surprise ourselves when we look back at the way it was when we first embarked upon this approach. When we first got started, we referred to it as the SESSION model. There have been numerous discussions regarding models like Anchor-Buoy, Session, and hybrid models. Ray Cologon authored a white paper that talked to this subject. Even though the framework relies heavily on the SESSION model, it also allows for a great deal of flexibility.

The one guiding principal of the FOCUS framework is that the layout of the graph and the way table occurrences are implemented, is from a functional perspective, instead of an entity or anchor perspective. This is a key part of what makes FOCUS what it is. We'll describe that in much more detail when we look at the graph and what each of the 3 letter abbreviations mean.

We also enjoy the freedom and flexibility that we have with FileMaker, especially with FileMaker 10's features like saved finds and the customized tool bar. However because of our reliance on our UI, and upon 'defining a focus', we have to sometimes hide the tool bar when it doesn't make sense within our framework. Where appropriate and possible, we do allow FileMaker 10 native features to show through.

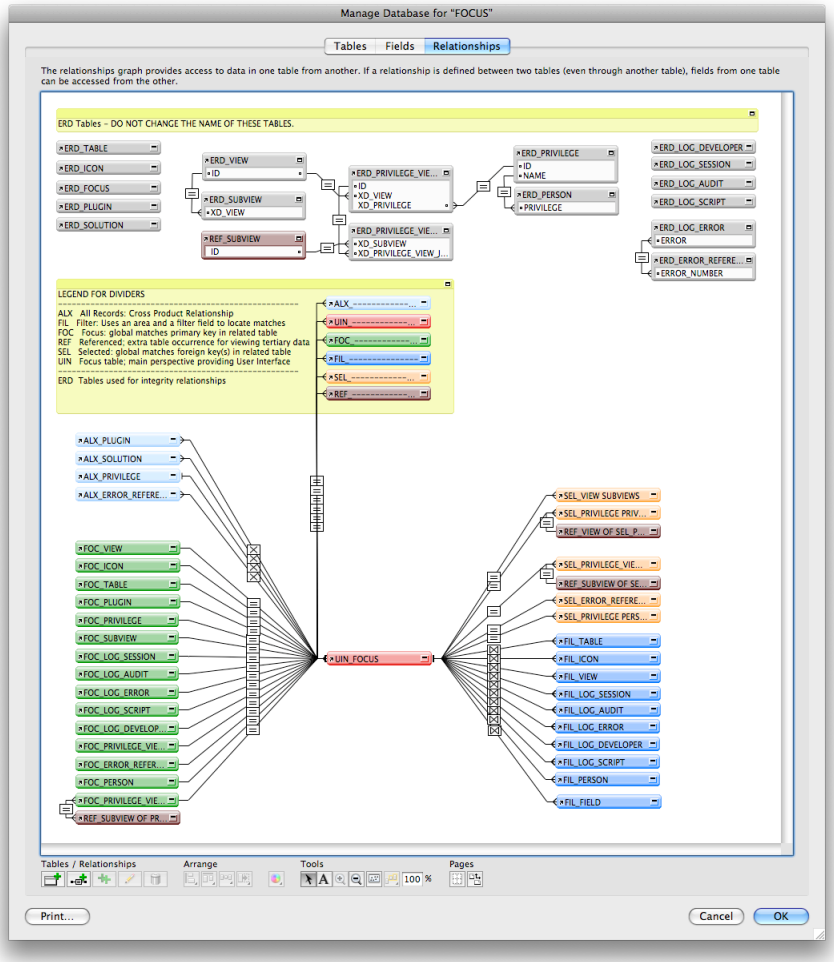
There are many approaches to designing and building FileMaker solutions. In our work, FOCUS has proven to be efficient and effective. FOCUS has also provided us with another very important aspect – that of shared responsibility and standardized approaches to FileMaker development. The framework can be improved, but we feel this is a great start – having had many years of real work experience behind us.

We are happy to share this approach with you and also know that this is not the be all and end all. As stated, FOCUS is a continually evolving framework. We look forward to hearing your feedback and suggestions. With your contributions, we'll make it even better.

Functional Approach

The main guiding principal in regard to FOCUS is that of "Function before Form". I think this is a great way of understanding how we work with most of the UI table occurrences in FOCUS. Let's have a look.

Here is a picture of the Graph in the FOCUS framework. Note that there are 2 main groups. You can define more groups if you see fit, but for now our 2 groups represented here are ERD (denotes entity relationship diagram), and UIN (user interface).



You will also see that we have 3 letter abbreviations that prefix each of our tables. There is a legend and it is also color-coded. Once you understand the approach, feel free to create your own 3 letter abbreviations.

The legend describes the *functional* behavior of that 3 letter group. So for example, look at tables that begin with `ALX` (All Records: Cross Product Relationship). Many of the interface layouts are derived from the context of `UIN_FOCUS`. If you put a

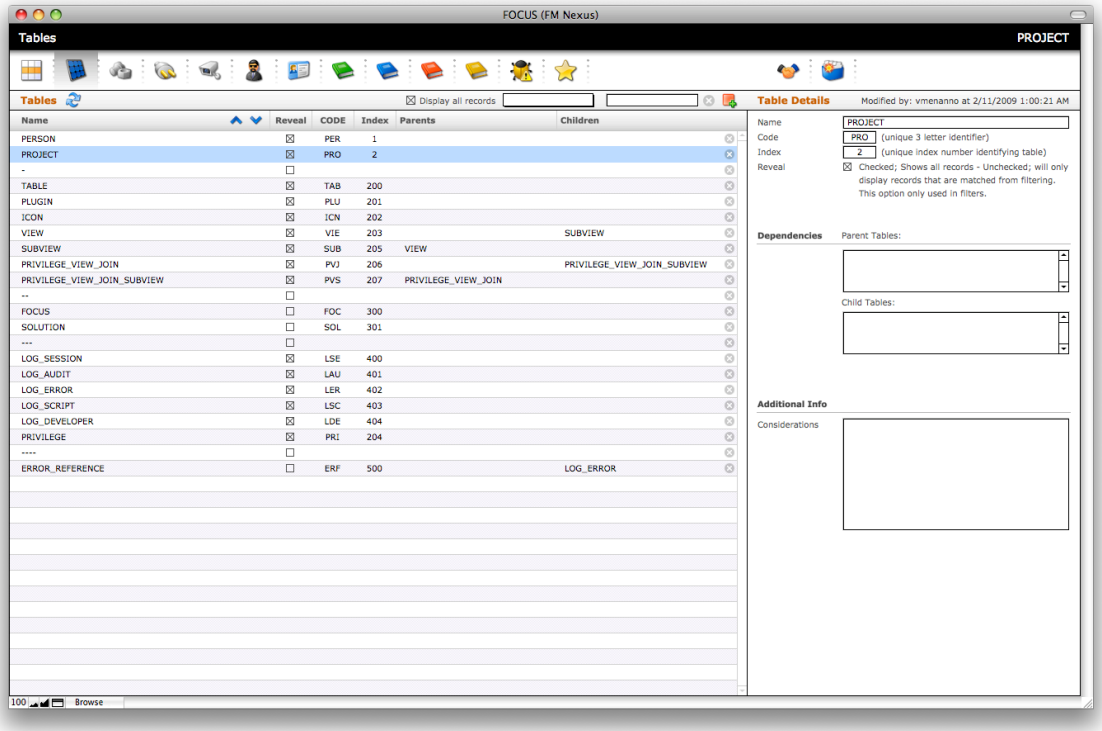
portal on the layout based on ALX_PLUGIN (light blue), then you would see all the records in the PLUGIN table.

Here's another example:

Let's look at the FOC_PLUGIN (in green) table occurrence. This relationship is based on GT_ID_FOCUS (in the FOCUS table) = ID (in the PLUGIN table). The GT_ID_FOCUS is a multi-key field, so if a field matches an ID, then we have a 'focus' on a specific record. Our keys are not only unique by NIC, RecID, and Timestamp, but also with their own unique 3 letter prefix (within the entire file). If you don't understand this instantly, don't worry too much, because it will become clearer as you start using FOCUS.

Adding a Table

FOCUS relies on the 3 letter abbreviations and a unique number for each table (kind of like you would have in an accounting program). You need to enter this information into the TABLE table. I know it looks like I just made a typo, but that is correct. The Table tab is the second icon from the left (the thing that looks like a solar panel). To create a record, click the red square with a green plus sign. This will create a new record in the TABLE table. Next, enter a 3 letter abbreviation and a unique index number for this table.



So let's say you were to add a new table to this database – before you get the urge to dive headfirst into the define database dialog, create your record in the `TABLE` table. Once you know you have a valid 3 letter code, then go ahead and add your new table into the schema of your database. In this case, we are ready to add a new `PROJECT` table.

All table names, field names, and table occurrence names are UPPERCASE. Each table will have a primary key field named `ID` with the following calculation (note 3 letter prefix at the beginning of the calculation). This `ID` creation was inspired by some research that Ray Cologon did to generate unique ids (so credit goes to him).

```
"PRO-" & Upper ( NIC_CREATED ) & "-" & Get ( RecordID ) &
"-" & GetAsNumber ( Get ( CurrentHostTimeStamp ) )
```

NOTE: In theory, this approach can allow for users to take offline copies of the database and have users create data and merge data back into the solution without any record conflicts and keeping each record uniquely identified. This has not yet been fully tested and used in a production environment.

Adding a Table to Schema

Now you are ready to add your table to your schema. Create your new table and copy the fields from another table. Listed below are the main fields that are in each table.

ID	Text	Indexed, Auto-enter Calculation replaces existing value, Can't Modify Auto, Always Validate, Unique, Allow Override
-	Text	
RECORD_ID	Text	Auto-enter Calculation replaces existing value, Can't Modify Auto
TIMESTAMP_CREATED	Timestamp	Auto-enter Calculation replaces existing value, Can't Modify Auto
CREATED_BY	Text	Auto-enter Calculation replaces existing value, Can't Modify Auto
TIMESTAMP_MODIFIED	Timestamp	Auto-enter Calculation replaces existing value, Can't Modify Auto
MODIFIED_BY	Text	Auto-enter Calculation replaces existing value, Can't Modify Auto
PRIVILEGE_SET_NAME	Text	Auto-enter Calculation replaces existing value, Can't Modify Auto
NIC_CREATED	Text	Auto-enter Calculation, Can't Modify Auto
--	Text	
NAME	Text	

Now that you have added your new table, you will also see its table occurrence on your graph. Rename that table occurrence with an "`ERD_`" prefix. Save your work.

You should now have new layout for this new table. Rename the layout to "# " (yes that is a space after the # symbol). With this new layout, you can go into browse mode and even start entering data (not so fast sparky!).

Adding a Great User Interface

Let's show you how to create a much better user experience to enter data.

<Watch a short movie>

Creating Records

FOCUS has a couple of ways to create new records.

Let's say you have a parent record that does not have any ancestor records – in this case, a project record. We use a generic record creation script when creating records that don't have any ancestors. Call the script "`b create record ( table )`" and in this case, pass "`PROJECT`" as the parameter for the table name. We'll need to modify the "`b create record ( table )`" script every so slightly to include "`PROJECT`" as one of the tables to watch out for. It's that easy. Here is what the `IF Statement` might look like when you are done adding the exception for the `PROJECT` table.

```
$_table = "VIEW" or
$_table = "PLUGIN" or
$_table = "LOG_DEVELOPER" or
$_table = "PROJECT"
```

The other method of creating new records would be to have the script verify that all systems are go to create a new project. So for example, let's say we had a table for tasks pertaining to a project. When the user tries to create a new task, the system ensures (before we create a new task record), that a project record exists and is selected.

Highlighting Records

Highlighting has improved over the years. We now use a very light-weight text object which fits exactly inside the portal row and has the following conditional formatting calculation.

```
PatternCount ( UIN_FOCUS::GT_ID_FOCUS ; ALX_PROJECT::ID )
```

If the above formula is true, then we change the fill color on the text object. Otherwise, we do nothing. Thus we achieved a highlight without the need of a field.

Highlighting is normally achieved by running the script:

```
b focus ( id ; -object ; -refresh )
```

Without getting too detailed at this time, basically the `GT_ID_FOCUS` field can hold a unique key for each one of the tables in the solution. Mapping that field to the `ID` field in each table provides us with a focus on that individual record.

Note: –object and –refresh are optional parameters.

Editing Data

Editing data can be achieved in a number of ways. Normally we show enterable fields that come from the perspective of a FOCUS table occurrence.

So let's say you wanted to edit the `PROJECT::NAME` field.

We would put the following field on the layout

`FOC_PROJECT::NAME`

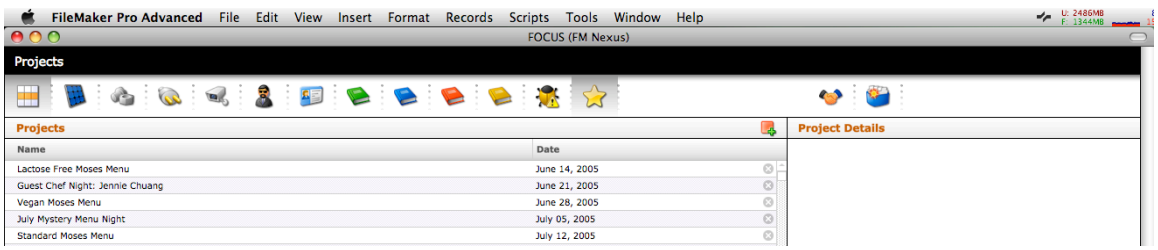
This is an enterable field with some visual indication that the field can be entered into.

If a field is needed only to display information, we might use a merge field or a non-enterable field.

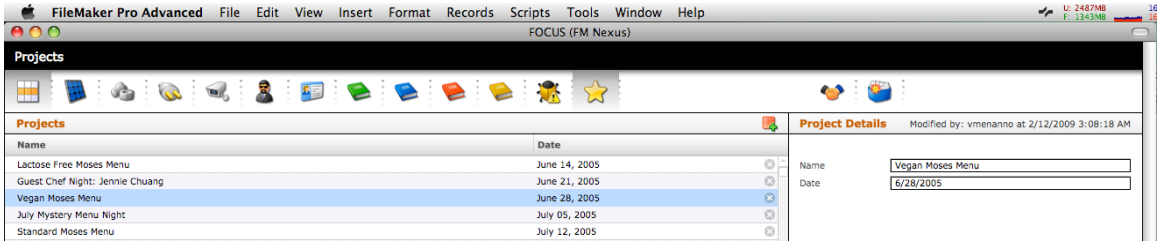
Masking techniques

We can mask over a number of different objects by putting a 1 row portal with no border, based on the `FOC = FOCUS` table occurrence for the data being modified. In the images that follow, the invisible portal row is on the right hand side.

In the case of the layout that displays a list of projects, if no project is selected, then no matching `ID` is present in `GT_ID_FOCUS`, and as such no 'focus'.



Once the user has highlighted a row in the portal, the focus is evident (to the right). See the illustration below:



Another masking technique employed is that of putting a text object with a single space in it, and attaching a conditional formatting calculation to the object.

Adding an FM Spotlight Filter

We like to call it FileMaker Spotlight! The great thing about FOCUS is that it has some great filtering examples already built in, and it has a very structured way of adding filtering.

Filtering is also a great way to ensure optimal database performance. In FOCUS, we also provide a 'reveal flag' for the table. What this flag allows you to do is to reveal only those records that match your filter, by constraining data to match your criteria. If the reveal flag is turned off, match records will start showing up only when you type in some values into the filter field. This is similar to the behavior you see in Apple Directory. In the case of a few hundred records probably not necessary, but in the case of thousands of records, you might want to use this feature.

In the PROJECT table add the following calculation field:

```
CT_FILTER (CT = Calculation, Result TEXT)
```

With the following calculation:

```
BuildIndex ( NAME ; "Name" )

BuildIndex is a custom function and it takes 2 parameters.

BuildIndex (fieldContents; fieldname)

fieldContents & ¶ &
fieldName & " " & fieldContents & ¶ &
IndexSpaces ( fieldContents ) & ¶ &
PermutateItem ( fieldName & " " ; IndexSpaces (
fieldContents ) ; 1 )
```

The field that you want to have broken out for filtering and the name of the field – this is particularly useful when you want to add more than one field into your filtering. For example if we added a project STATUS field, then our calculation in CT_FILTER would look like this:

```
BuildIndex ( NAME ; "Name" ) &
BuildIndex ( STATUS ; "Status" )
```

Something else to note when filtering in this way is that this field will then hold the index, and as such you don't need to index the NAME or STATUS field.

Now that you have a filter field, we'll need to add fields in the FOCUS table and create a relationship that serves as our filtered portal of PROJECTS. The easiest way to accomplish this is to simply duplicate one of the other pair of fields and rename them to include the name of the table (in this case PROJECT).

```
CT_FILTER_LOW_PROJECT
CT_FILTER_HIGH_PROJECT
```

The calculation for the *low* value is the following:

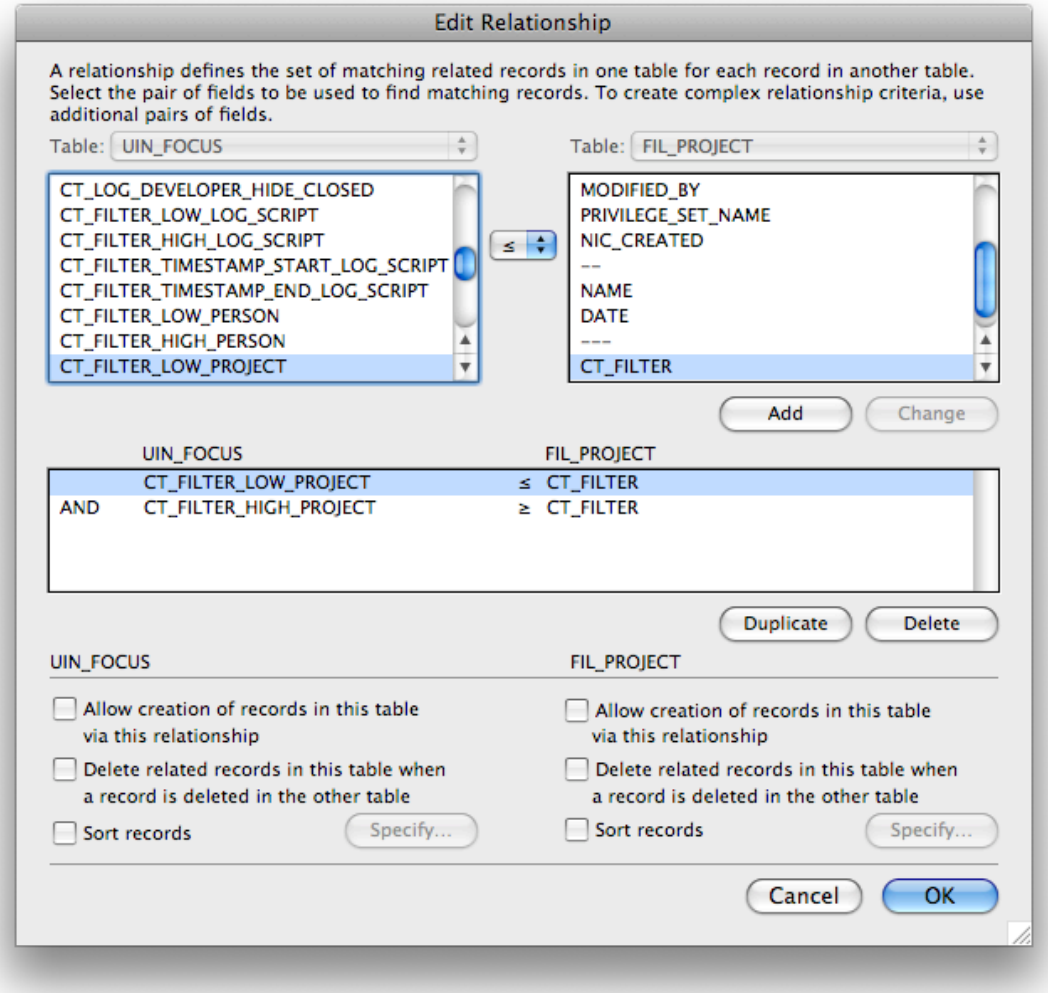
```
FilterLow ( ExtractTableName ( GetFieldName ( Self ) ) )
```

The calculation for the *high* value is the following:
Won't go into too much detail here about this calculation – all you have to do though is change the table name and the rest is handled by FOCUS.

```
Let ( [
  #_table = "PROJECT" ;
  #_index = TableIndex ( #_table ) ;
  #_filter = UIN_FOCUS::GT_FILTER [ #_index ] ;
  #_reveal = GN_FILTER_FLAG_REVEAL [ #_index ] ;
  #_field = GT_FILTER_FIELD [ #_index ]
  ] ;
  Case (
    #_reveal = 1; // All records are showing
    If (
      IsEmpty ( #_field ) or #_field = FilterFieldDefault;
      #_filter & FilterLimitHigh;
      If ( IsEmpty ( #_filter ); #_filter & FilterLimitHigh
; #_field & " " & #_filter & FilterLimitHigh )
    );
    #_reveal = 0; // No records are showing
    If (
      IsEmpty ( #_field ) or #_field = FilterFieldDefault;
      If ( IsEmpty ( #_filter ); ""; #_filter &
FilterLimitHigh );
```

```
      If ( IsEmpty ( #_filter ); ""; #_field & " " &
#_filter & FilterLimitHigh)
    )
  )
)
```

Once you have your fields defined, you'll want to add the table occurrence to the graph. In this case, you'll be adding `FIL_PROJECT` and creating the following relationship between the `PROJECT` table and the `FOCUS` table.



<Watch a movie to see how Filtering is put together>

Scripts and Parameters

In FOCUS we take advantage of FileMaker script groups to organize scripts into logical sections. We also have a convention where if a script is called by another script, that the script name be preceded with a period. For example the script "`b create record ( table )`" is a script that is attached to a button on the layout and when it is called it may call the following script "`. create generic ( table )`"

Another thing that we strive for with FOCUS is the ability to black box all of our scripts such that they can be easily copied or run from any context. As long as the script receives the parameters it is looking for, the script can be called from any place.

Parameters are a big part of FOCUS and we use a parameter passing technique that sixfriedrice came up with. Basically, it is a name value pair encoding that allows us to easily send a script one or more parameters, and then have the script unpack them. The two custom functions that we use to set parameter and get parameters are:

```
GetParameter

#( key, value )

// SetParameter ( key ; value )
// Returns a dictionary entry with name and value
// SetParameter("ID", 2) & SetParameter("NAME", "bill")

"<:" & key & ":=" & "TEXT" & ":=" & Substitute ( value ; [
"=" ; "/=" ] ; [ ":" ; "/:" ] ; [ ">" ; "/>" ] ; [ "<" ;
"/<" ] ) & ":>"
```

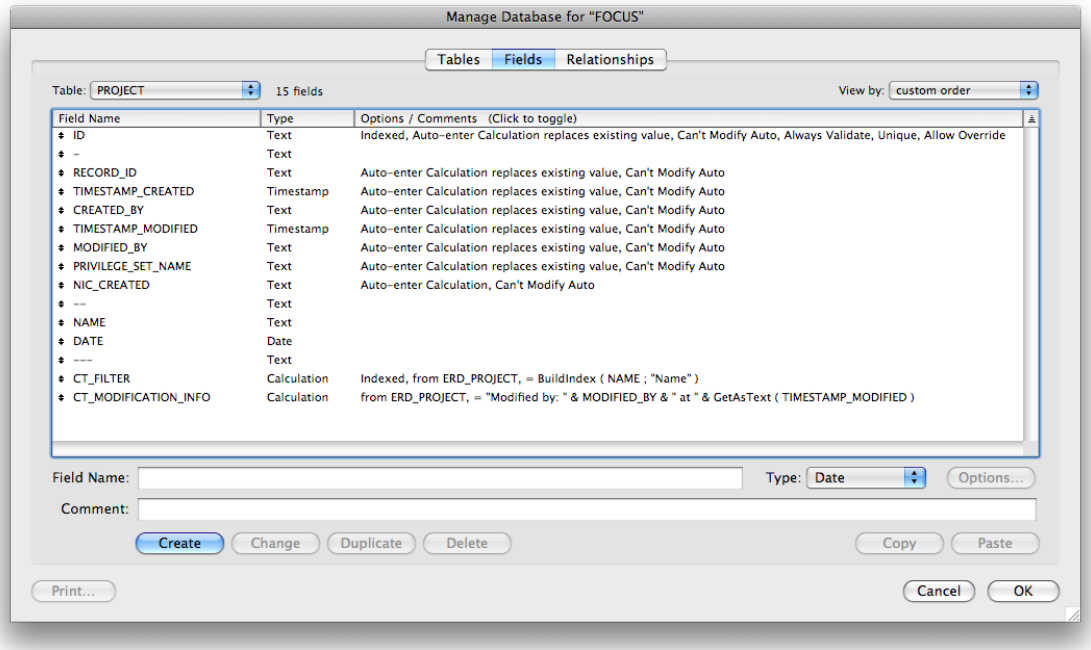
```
SetParameter

#P ( name )

// GetParameter ( name )
// Assuming the script parameter contains a dict, returns
the value of key 'name' in the script parameter
// GetParameter ("ID")

DictGet ( Get ( ScriptParameter ) ; key )
```

These two custom functions are incredibly useful, reliable and powerful. They are used not only in scripts to move parameters from one script to the next, but also on buttons, triggers and custom menus. Basically anywhere you need to send a parameter to a script, this is the method used.

Organization and Groups

We wish that FileMaker would have groups in other places similar to script groups. But because it does not, you'll find that we try to create groups of things in various places like custom functions, value lists, layouts, etc.

In define database dialog, you'll also see groups. In some tables there are more groups than in for other tables. This picture is a great example of the 4 groups that we normally have all of our tables. At the very top we keep all the IDs together (primary and foreign). Then comes a list of fields used for meta data – such things as by whom and when the record was created or modified. We then include a group of fields for the actual data that we will be interacting with. Lastly, the grouping at the bottom contains the list of calculation fields.



Authors

FOCUS was originally developed by Vincenzo Menanno (FM Nexus) and significant contributions from Will M. Baker (Beezwax Datatools).

We also would like to thank the silent contributors who have contributed indirectly. People like Geoff Coffey and Ray Cologon who, with their contributions to the FileMaker community, have also ended up inspiring us to include their ideas, and some times their actual code.

Terms of Use

FOCUS is freely provided. However it does currently require the use of 2 plug-ins. The FM Nexus Utilities plug-in is free, but the Simple Dialog plug-in requires a license – we acquired a special 1 hour trial license so that we would be able to share the file in an unlocked state.

The other requirement if you decide to use the framework, is that you make visible the fact that you are using the FOCUS framework and provide acknowledgements to the authors mentioned above.