Field naming

Field ordering and organization within Manage Database

The standards documented here do not use any special field prefixing of cryptic or shorthand characters. You won't find a prefix such as gkt_SomeField, where **gkt** may stand for **Global Key Text** field. Instead, these standards focus on descriptive field naming with implied use or type. For example, **customerPicture** or **productImage** clearly indicates the field type of container. These standards do not enforce (but may suggest) the use of certain keywords, such as **Picture** vs. **Image** for field names. For field types, such as key fields vs. globals, you will find strict guidelines.

Field sorting order in Manage Database > Fields dialog

Please note that the standards covered within this section assume you will be organizing and arranging your Relationship graph based on an explicit and controlled field order. This is accomplished using the **View by: custom order** setting within the Fields tab of Manage Database. This should be set *before* the file is hosted. See the pre-hosting checklist for more information.

Switching to "field name" or "field type"

Using the **custom order** setting comes with the condition that you ONLY drag your fields into the explicit desired order when the **View by:** setting is **custom order**. If you temporarily switch to **field name** order and then drag one single field, you will replace your previous **custom order** sorting. YES, this may require forming a new habit, however, the advantages to a known and fixed field order lends to quicker development time based on positional familiarity. A good example would be having all apps on an iPhone just rearrange themselves based on alpha order. Your mind maps positional locations of things and this makes things faster.

Key fields

- **Primary keys** in each table are simply, **and always**, named "**id**" – Fields are identified by FileMaker inherently by the Table » Occurrence::id nomenclature. (e.g. Schedule » People::id)
- **Foreign keys** within any table are named using an underscore followed by the exact same Title case version of their table name. (e.g **id_Tablename**).
  The underscore within the field name distinguishes a foreign key from the camelCase field names

`id_People`	good
`f_personID`	bad
`id_Event`	good
`kf_Event`	bad

Serial Numbers vs. UUIDs

While not a required standard, the majority of contributors to this standard suggest using universally unique identifiers (UUIDs) for primary key values. This avoids some common issues with FileMaker's default serial value option. See the Best Practices for key values section for information about using UUIDs.

- **Constant keys** should be prefixed with the keword "**key**_" and follow the format used by GLOBAL fields. Because these types of fields are not primary keys, yet they are used for utility relationships, they can be interpreted as foreign keys of a special type.

`key_BOOLEAN`	good
`zkf_ConstantOne`	bad
`key_ALLCUSTOMERS`	good

  Note: try to name the key functionally relevant and easily identifiable, for example **key_ALLCUSTOMERS** may indicate an unstored constant calculation derived from another relationship using a List() function for the purpose of a cartesian relationship

- **Multi-key key** fields are a feature unique to FileMaker Pro. They provide a feature which allows for the implementation of pseudo-schema (described later) or for a purely utilitarian feature of being able to drive data display within FileMaker Layouts - known as the UI of a solution. Because these multi-key values, which are simply what FileMaker calls a List (see the List() function) can be anything from a list of serial number values to UUID values to a list of concatenated keys, it becomes useful to easily identify such use of these fields. The two prefixes used to identify these fields are **idList**_ and **keys**_. Choosing one over the other can be a bit subjective, but follows these guidelines. If the field storing multi-key values relates to the structure, commonly known as the schema, for a solution and uses id values, then idList_ is preferred. The keyword "List", within idList_ implies the field stores multiple values. When the field is used for the purpose of utility, then the prefix keys_ is preferred. The keys_ prefix implies multiple values due to it being plural as opposed to the **Constant key**_ prefix above. When used in the context of a relationship, the trailing portion of the field name is typically the Tablename to which the foreign keys are being related to.

idList_Customers	good
`keys_Participants`	good
keys_INVOICES	good

In the above examples, idList_Customers might be used to maintain a pseudo-schema relationship between a customers table and an Invoices table. This type of use would be in place of a normal Join table and is a subjective decision based on the criteria and needs of the solution. keys_Participants may represent a record local list of id values creating a many-to-many relationship from one record to many records within a Participants table. The keys_INVOICES field is clearly a global field used for the purpose of holding any number of multi-key values which may be id values, serial values or concatenated keys for the purpose of data display within a portal.

Pseudo-schema

When considering the principles behind database normalization, a typical approach to structuring data is to use a join table. However, in the world of FileMaker, a relationship between two tables can take advantage of a feature known as a multi-key field. This field can store more than one single key value. As stated on this web site, this feature is somewhat unique to FileMaker and does not follow the principles of data normalization. Strictly within the context of FileMaker, however, using this method of relating data is perfectly viable and is being called pseudo-schema here because of its uniqueness. The most common use of multi-key fields is for the purpose of driving data display within the user interface. It is possible, however, to maintain, as part of the solution schema, multi-key values within fields for the purpose of data structure. Use the above guidelines to direct your use of these prefixes.

- **Shared keys** which are fields that are foreign in nature, yet do not correlate to any one specific table, should use a fixed name within all tables throughout the solution. These standards suggest the use of the name **id_Any** where **Any** can be the primary key from any table. One example of a **shared key** would be a foreign key within a shared Notes table. Where the **id_Any** field could potentially hold the primary key value for tables named People, Events, Invoices, Estimates, etc. In this scenario, the Notes table is used to store notes for many other tables.

Developer note: the use of Shared keys is typically a practice which utilizes additional predicates within relationships, such as a noteTable field used to qualify which table a specific note key belongs to.

Data fields

- Data fields, those used for solution data, use **lowerCamelCase** starting with a lowercase letter. Starting with lower case differentiates the string from function calls which are **Title Case**.
  No spaces or underscores

`customerFirstName`	good
`Customer First Name`	bad
`CustomerFirstName`	bad

Summary fields

- Summary fields are prefixed by the word "**summary**" and their return type before their identity. This facilitates quick and easy recognition.

`summaryCountMembers`	good
`summaryMemberCount`	bad
`memberCount`	bad
`summaryMaxId_event`	good
`summaryMinAmountSpent`	good
`summaryAvgPaid`	good
`summaryTotalUsers`	good
`summaryStdDevWhatever`	good
`summaryFracTotalMetrics`	good

Note the use of the underscore above, it indicates a summary of a foreign key (covered below)

Global fields

- All GLOBAL fields are **UPPERCASE** and one single word. This easily differentiates global data from all other types.

`CREATE`	good

EVENTFILTER	good
CUSTOMERSEARCH	good
CLIPBOARD	good
g_GlobalFieldName	bad

Underscore allowance

Based on certain situations when global fields do not read well as a single word, you can optionally using the underscore as part of these standards. So **GL OBALLANGUAGE** would become **GLOBAL_LANGUAGE**. However, it's suggested you attempt to alter your word choices to attempt to use the single word approach.

Unstored calculation fields

- Unstored calculated fields are prefixed with the word "unstored," and that prefix is separated from the rest of the field name with an underscore. Always try to use descriptive adjectives or actual nouns to help identify the field type (e.g. Count in userEventCount indicates numerical data type)

unstored_userEventCount	good
uc_EventCount	bad
unstored_invoiceStatusImage	good
%invoiceStatus	bad

A discussion of the factors that lead to this decision is available on the fmstandards Google Groups forum.

Default Auto-Enter fields

- Default auto-enter options provided by FileMaker are grouped by their prefix and identified by their respective name. These include the following defaults

```
creationDate
creationTime
creationAccountName
creationTimestamp
creationUser
modificationDate
modificationTime
modificationAccountName
modificationTimestamp
modificationUser
```

Note that not all fields are required. These are the suggested names.

- Two additional fields suggested by these standards are **creationHostTimestamp** and **modificationHostTimestamp** as optional default fields. The method of adding these fields to your database are covered under the Host Timestamp fields section.

```
creationHostTimestamp
modificationHostTimestamp
```