FileMaker Sync Guide

Best practices for building your own system

Table of Contents

Introduction

With FileMaker Go for iPad or iPhone, data in FileMaker systems can be sent out on the road, allowing users to work independently from hosted FileMaker Pro files. In addition, data collected in the field can be added to the hosted system. If remote users have a full-time network connection with sufficient bandwidth, they may work directly with the hosted system and will have no need to sync. But for remote FileMaker Go users who need the freedom to move without relying on a network connection, syncing a deployed system to a hosted system is an essential part of a solution designed specifically for them.

Syncing data between the two systems may be as simple as importing records from one file to another, or as complex as incremental record-by-record updates based on business rules. This document discusses the two separate but interrelated components of syncing—record exchange and record processing. While the focus of this paper is on syncing between FileMaker Go files and hosted FileMaker Pro files, the concepts may also be applied to syncing between files used solely by FileMaker Pro users (for instance, syncing a FileMaker system deployed on a laptop with a hosted FileMaker system).

This document begins with an exploration of sync strategies: Where should sync start? How will the sync process determine if the hosted file is available? What should happen if the sync process is interrupted?

Following these initial concepts is a discussion of network connection and disconnection, how scripts resume if connection is interrupted, and when, how and why to close hosted files on the iOS device. Next, record processing is covered, including identifying records for sync and when and where record processing should take place. Additional points are raised about Issues that should be considered to determine which records should be updated, and in what way, in both by hosted files and files deployed on FileMaker Go.

The next section provides an outline of the steps involved in a sync process.

The final section of this document is a detailed exploration of a sync process. This section features the use of native FileMaker functions and is founded on the principle that the sync process should rely on a network connection for as short a time as possible. The section includes sample scripts and advice for structuring tables, relationships and scripts.

This document explores high-level concepts and offers detailed technical guidance for users who are interested in building their own FileMaker systems, and offers adventurous beginners an introduction to the power and potential of FileMaker. This paper may also shed light on the concepts behind commercially available sync solutions. It assumes that the reader is already familiar with many of the features of FileMaker Pro and has a good understanding of scripts.

Sync Concepts

Syn•chro•nize (verb): To coordinate or combine.

Sync has several meanings, referring to different processes in different scenarios. For users who collect a set of work orders or assignments at the beginning of the day, synchronization may simply involve downloading records assigned to them from a hosted file to their FileMaker Go custom business solutions at the day's start, and, at the end of the day, uploading notes made on those work orders from FileMaker Go to the hosted file. Assuming that the system is designed appropriately, these uploaded notes will automatically be associated with each user's work orders in the hosted system. The following figure illustrates this type of sync.

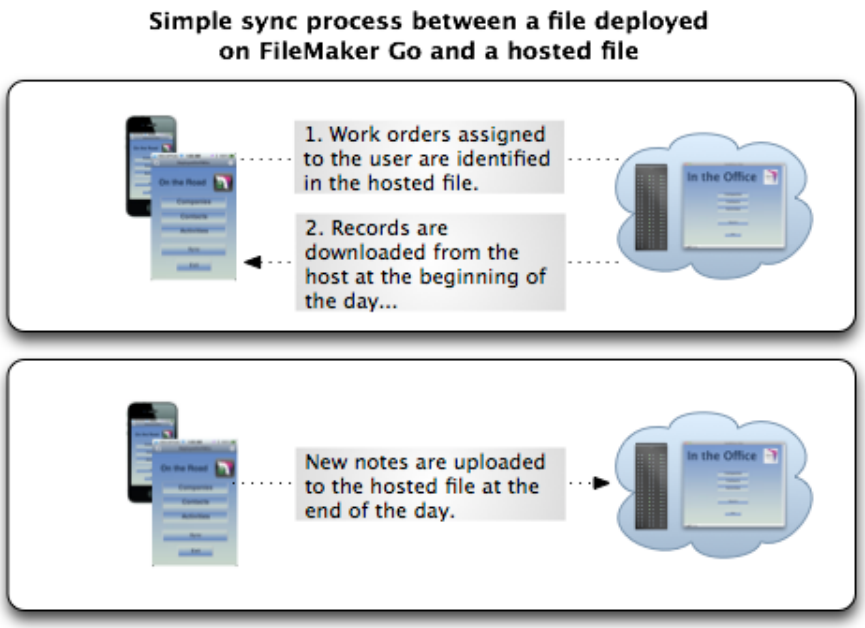

Simple sync process between a file deployed on FileMaker Go and a hosted file
1. Work orders assigned to the user are identified in the hosted file.
2. Records are downloaded from the host at the beginning of the day...
New notes are uploaded to the hosted file at the end of the day.
In the Office
On the Road

Figure 1: A simple type of sync.

For users who need more timely information, and where changes may be occurring during the same period on both the hosted solution and the FileMaker Go solution, sync could mean coordinating record data to ensure that the same information is available in the hosted file and the file deployed with FileMaker Go. While the following graphic is simple, Step 2 requires the sync process to make record-by-record decisions in order to selectively apply updates in one or both systems:

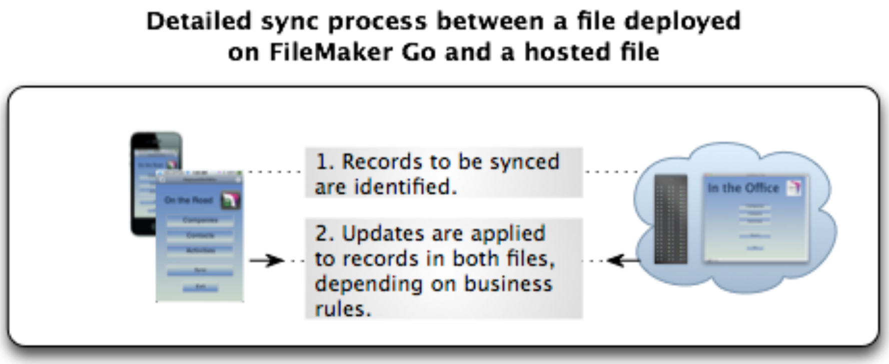

Detailed sync process between a file deployed on FileMaker Go and a hosted file
1. Records to be synced are identified.
2. Updates are applied to records in both files, depending on business rules.
In the Office
On the Road

Figure 2: A more complex type of sync, addressing business rules.

These sync scenarios include two files:

- **Deployed file:** A FileMaker file deployed on an iPad, iPhone or iPod Touch and run with FileMaker Go.
- **Hosted file:** A FileMaker file hosted by FileMaker Server or FileMaker Pro and run with FileMaker Pro on desktop or laptop computers or with FileMaker Go on an iOS device.

In addition, the sync scenarios discussed here separate the process into its two components: exchanging data and processing changes. Depending on business rules, changes may be processed as part of the data exchange. Separating the two functions allows one part of the sync procedure to be performed “off-line,” thereby minimizing the amount of time FileMaker Go will require a network connection. However, in cases where efficiently designed sync scripts are deemed to be as fast and reliable as necessary, the choice of whether to combine or separate data exchange and processing will be based on other factors. In general, it is better to have fewer “moving parts,” requiring the smallest amount of resources and time on the part of the iOS device—and viewing the process as a whole, this reduces the amount of time remote users will have to wait for sync to complete.

Exchanging Data

One of the key components of any sync process is moving data between the files to be synchronized—e.g. the file deployed on FileMaker Go and the file hosted in FileMaker Server. The sync process may import records, use relationships managed by the sync script, or call scripts in a hosted file to create, modify or delete records. New records may be moved into their corresponding tables, and edits may be made to existing records. Alternatively, new and edited records may be moved into intermediate tables that keep them separate from the “live” tables until those records are addressed in a separate process.

However, not all data need to be exchanged (for example, data that have not been modified since the last sync). To identify records that should be included, your sync process should first look for:

- Records that have been deleted from either file since the last sync
- Records modified since the last sync
- New records (created in either file since the last sync)

Processing Records

Whether live records are updated during data exchange or in a separate process, the same concepts apply: Records with creation dates prior to the last sync that are absent from either file should be deleted, records that exist in both files should be modified if they have been changed since the last sync, and records in either file that have a creation date since the last sync should be created in the other file.

One potentially challenging aspect of sync is priority of modifications. If a record in the hosted file was modified in the morning, and the same record in the deployed file was modified in the afternoon, which modification should take precedence? The answer to this question, and others like it (such as whether users should be allowed to delete records in the solution) depends on business rules for the system. For further discussion, see the “Business Rules” section in *Developing Your Sync Strategy* below.

FileMaker Scripts

While users may manually import records into FileMaker Go and FileMaker Pro files, due to its complexity, the process is best handled using a FileMaker script. The script may be attached to a button that remote users select to initiate the sync. If record processing is required in the hosted file subsequent to data exchange, another script may be scheduled or initiated on FileMaker Server by the deployed file at the appropriate point in its sync script. Alternatively, a FileMaker system can be designed and deployed to run scripts that act on the hosted file, in order to perform this task.

FileMaker scripts are created in FileMaker database files using FileMaker Pro or FileMaker Pro Advanced. They may then be run in files deployed with FileMaker Go, files accessed in FileMaker Pro, and files hosted in FileMaker Server. To learn more about writing scripts with FileMaker Pro, see the resources available from the FileMaker Technical Network site as well as Module 6 of the *FileMaker Training Series* (links to both are in the Resources section).

Functions and Roles of FileMaker Components

This document assumes that you will be syncing records between a deployed file running on FileMaker Go and a hosted file on FileMaker Server, as described above. (As mentioned in the introduction, these sync concepts also apply to sync performed between two FileMaker files.)

Syncing between a FileMaker Go solution and a hosted solution involves three components—FileMaker Pro, FileMaker Server and FileMaker Go. The following figure shows how the components work together:

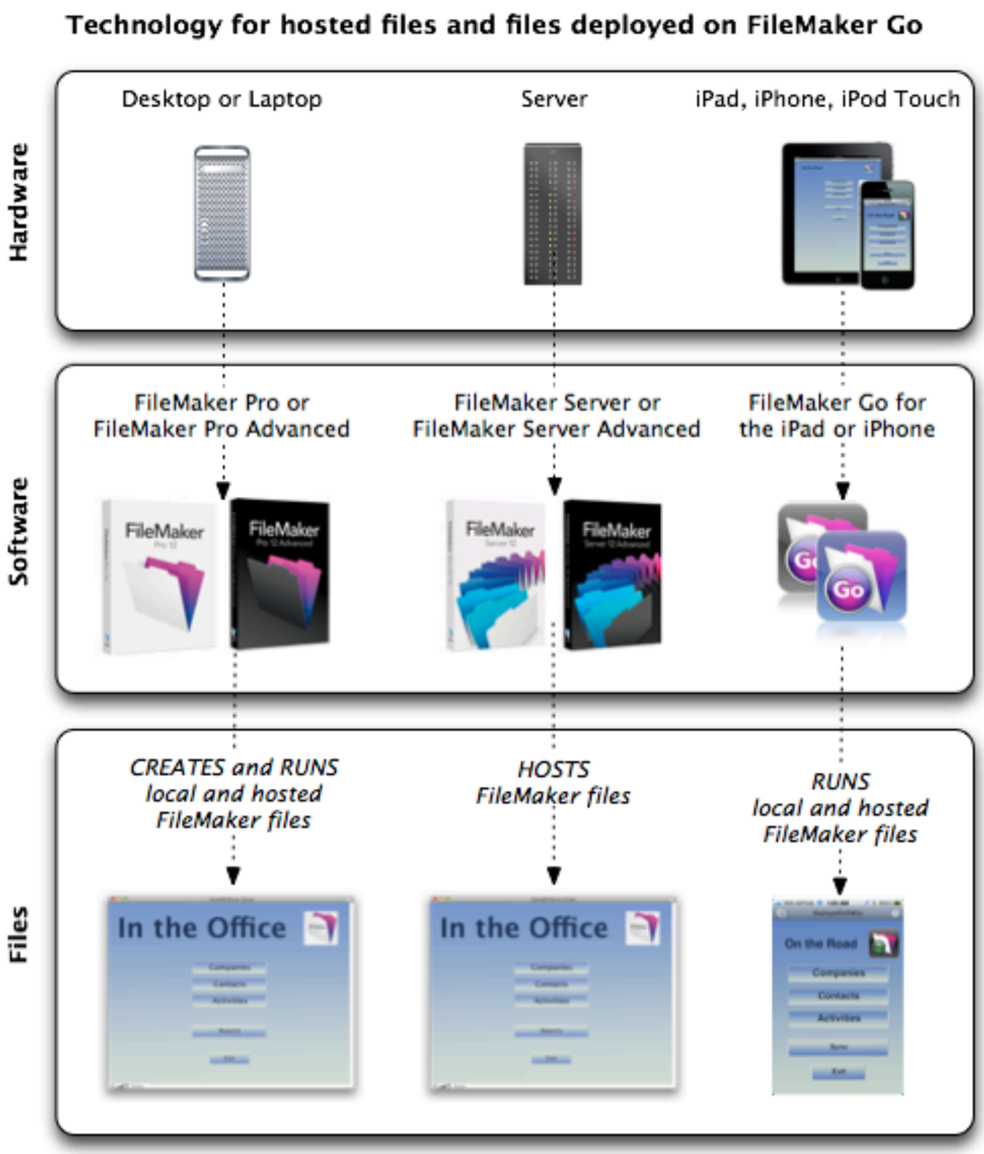

Technology for hosted files and files deployed on FileMaker Go
Hardware
Software
Files
Desktop or Laptop
Server
iPad, iPhone, iPod Touch
FileMaker Pro or FileMaker Pro Advanced
FileMaker Server or FileMaker Server Advanced
FileMaker Go for the iPad or iPhone
CREATES and RUNS local and hosted FileMaker files
HOSTS FileMaker files
RUNS local and hosted FileMaker files
In the Office
In the Office
On the Road

Figure 3: The components required to create, host and run FileMaker files.

Developing Your Sync Strategy

In developing your sync strategy, you may wish to consider the following factors:

- As noted above, local files opened with FileMaker Go may have file references to hosted files, but not vice versa. Therefore, it is possible to have relationships in files deployed on FileMaker Go to hosted files or files on the same device, and to call scripts in those other files, but it is not possible to create relationships to or call scripts in FileMaker Go files from files that reside outside of the iOS device. However, hosted files opened on the iOS device can use the fmp URL protocol to open files and perform scripts in files on the iOS device. (A number of documents about the fmp URL protocol are available on FileMaker's TechNet—see the Resources section at the end of this document.)
- Files opened on the iOS device may be closed if there are no table occurrences to them in any other files open on the device. Put another way, if any files open on the device include table

occurrences dependent on any other files (including hosted files) open on the device, those other files will not be closed even if the user manually closes the file windows.

- A script running in FileMaker Go that accesses a hosted file will be interrupted if the network connection is lost. Similarly, a FileMaker Go script will be interrupted if the iOS device's focus changes (e.g. if there is an incoming call or the user presses the home button).
- FileMaker Go will attempt to reconnect to hosted files and to resume an interrupted script when network connection is restored.
- Keeping track of the sync status (i.e. the timestamp of the last successful sync) allows subsequent sync processes to proceed efficiently (dealing only with records that have been added or changed since the preceding sync).
- Each user (or device) should generate unique primary keys for new records to ensure that records created on different devices will not have identical primary keys.
- Record-specific creation, modification and synchronisation timestamps will be useful for identifying records to be synced.

Planning From the iOS Device's Perspective

Since hosted files cannot reference files deployed on the iOS device, the deployed file must initiate the sync process. Consequently, any data to be moved to or from the hosted system via relationships or import processes must be managed by the deployed file or another file residing on the iOS device. Alternatively, data can be passed between files as delimited arrays within a script parameter and/or script result; however, detailed discussion of this approach is beyond the scope of this paper.

Helpfully, a file on the iOS device can have file references to the hosted file. Consequently, a "connector" file may be deployed on the device and used to move data between the hosted file and the deployed file, in which case the connector file will have table occurrences pointing to the hosted file AND the deployed file, while the deployed file will not include any table occurrences pointing to tables in the hosted file. The key benefit of using a connector file in this way is that upon closing the connector file at the conclusion of the sync process, it is then possible for the deployed file to also close the hosted file (whereas if the deployed file contained table occurrences referring to the hosted file, it would not be possible to close the hosted file without also closing the deployed file). Connector files are discussed in more detail in *The Sync Process Step By Step* section below.

With this concept of the deployed file opening and closing the hosted file and the connector file, following is an outline of the main steps suggested for the scripted procedure:

1. The deployed file runs a script to determine if the hosted file is available for sync.
2. If so, the deployed file opens the connector file.
3. The connector file identifies records in both the deployed file and the hosted file that require synchronization.
4. If any records are identified as candidates for synchronization, the connector file directs the process of adding new records, editing existing records and deleting records according to business rules. It does this by any of these methods:
    - Adding/editing/deleting records in both the deployed file and the hosted file.
    - Adding records to intermediate tables in the deployed file and the hosted file, then initiating post-processing script sequences in both files.
5. The deployed file closes the hosted file.

Additional considerations for scripts running in FileMaker Go:

When a script that allows user abort is in progress in a file on FileMaker Go, if the user taps the screen, FileMaker Go will ask if the user would like to continue the script. User abort is permitted by default in scripts unless Allow User Abort is set specifically to [Off]. Although Allow User Abort [Off] is not appropriate throughout every script run on iOS devices, it should be applied, at the very least, for the portion of a sync script that modifies record data, in order to preserve transactional integrity. If you would like to allow users to interrupt the sync process at other points, use Allow User Abort [Off] just before the steps that modify records, and Allow User Abort [On] just after. Alternatively, to prevent users from interrupting the sync process at any point, add Allow User Abort [Off] at the top of the scripts in the sync process discussed here. The Allow User Abort state will be passed on to sub-scripts, so you may choose not to include the step in a sub-script that will be called by another script that has Allow User Abort set to the appropriate state.

Security

The sync process will require access to create, edit and (contingent on business rules) delete records and data in both the hosted file and the deployed file. If the user's accounts in either file do not include any of these privileges, then calling a script in the target file that is set to run with full access privileges should perform the relevant actions. For example, a script designed to delete records in the hosted file can be set to run with full access privileges and configured to delete records whose primary keys are passed to it via a script parameter.

An additional security option is to set FileMaker Server to encrypt data passed between the database server and FileMaker clients. If encryption is required, your solution can use the Get(ConnectionState) function to confirm that secure socket layer (SSL) is being used for communication with the host.

Also, FileMaker's File Access feature allows you to authorize specific external files to work with the hosted file. This constraint allows you to control which files may access the hosted file, adding another layer of security to the sync process.

Finally, if the hosted system resides within a secure network, consider setting up an instance of FileMaker Server outside the secure network to act as an intermediary between iOS devices and the main host. In this scenario, the secure portion of the system can initiate contact with the portion outside the secure network, exchanging records to facilitate syncing by deployed files. For an example of this scenario, see the FileMaker webinar "FileMaker Go in Business: Austin Convention Center."

Network Connection

Network connection is key to sync—without it, sync is impossible. Because continuous network connection for an iOS device cannot be guaranteed, the sync process is vulnerable to interruption. Remote users will be connecting via wireless networks (on the same or different subnet from the host database server) or via available cell technology (e.g. 3G or 4G). None of these methods provide guaranteed bandwidth or continuity of connection.

Establishing Server Availability

To manage the possibility of connectivity loss, include frequent tests in your scripts to confirm that the hosted file is available. See the section "Resuming Sync if Interrupted" in *The Sync Process Step by Step* for information on structuring the scripts.

The first test to see if the hosted file is available should be included at the commencement of the sync process. Set Error Capture to [On] and use the Open File [] script step to call the hosted file. The result of Get(LastError) will indicate whether the file is available.

If the Hosted File is Unavailable

The hosted file may be unavailable because the file is not open. In this case, FileMaker Go will get a relatively quick response from FileMaker Server that the file is unavailable, and the result of Get(LastError) will be 100.

The hosted file may also be unavailable because there is no network connection to FileMaker Server. This may be because of one or more of these factors:

- The iOS device has no network connection
- The network via which the iOS device is communicating has no connection to the FileMaker Server network
- The network for FileMaker Server is unavailable
- FileMaker Server is not running

If there is no network connection to FileMaker Server, FileMaker Go will attempt to connect for up to approximately 1 minute and then will return 802 as the result of Get(LastError).

If the Network Connection is Lost

If network connection to a hosted file is lost (or if the file is closed on FileMaker Server) while the file is open on the iOS device, FileMaker Go will attempt to reconnect for a number of seconds, during which time the remote user will not be able to work with any FileMaker Go files. If connection is restored within this time period, FileMaker Go will attempt to reconnect to hosted files that were open before the disconnection. If the hosted files are available after network access returns, the first question is whether FileMaker Go will automatically re-authenticate (enter the user name and password previously entered for the file). But first…

If FileMaker Go Hibernates

Other ways connection to a hosted file may be interrupted include:

- The user hits the Home button on the iOS device
- The user switches to another app on the device using a four-finger swipe gesture
- A phone call comes in to an iPhone or a FaceTime call comes in to any iOS device
- The user puts the device to sleep
- The device goes to sleep automatically

In such cases, FileMaker Go is pushed into the background and is hibernated or suspended. When FileMaker Go is brought back to the foreground, it will attempt to reconnect to the hosted file. If the reauthentication period (if any) specified for the user's privilege set has been exceeded, it will reauthenticate with the hosted file. For more information, search "fmreauthenticate" in FileMaker Pro's Help.

Record Modifications after Disconnection

FileMaker Server handles record modification differently depending on the circumstances. While a record is being edited, either by a user or a script:

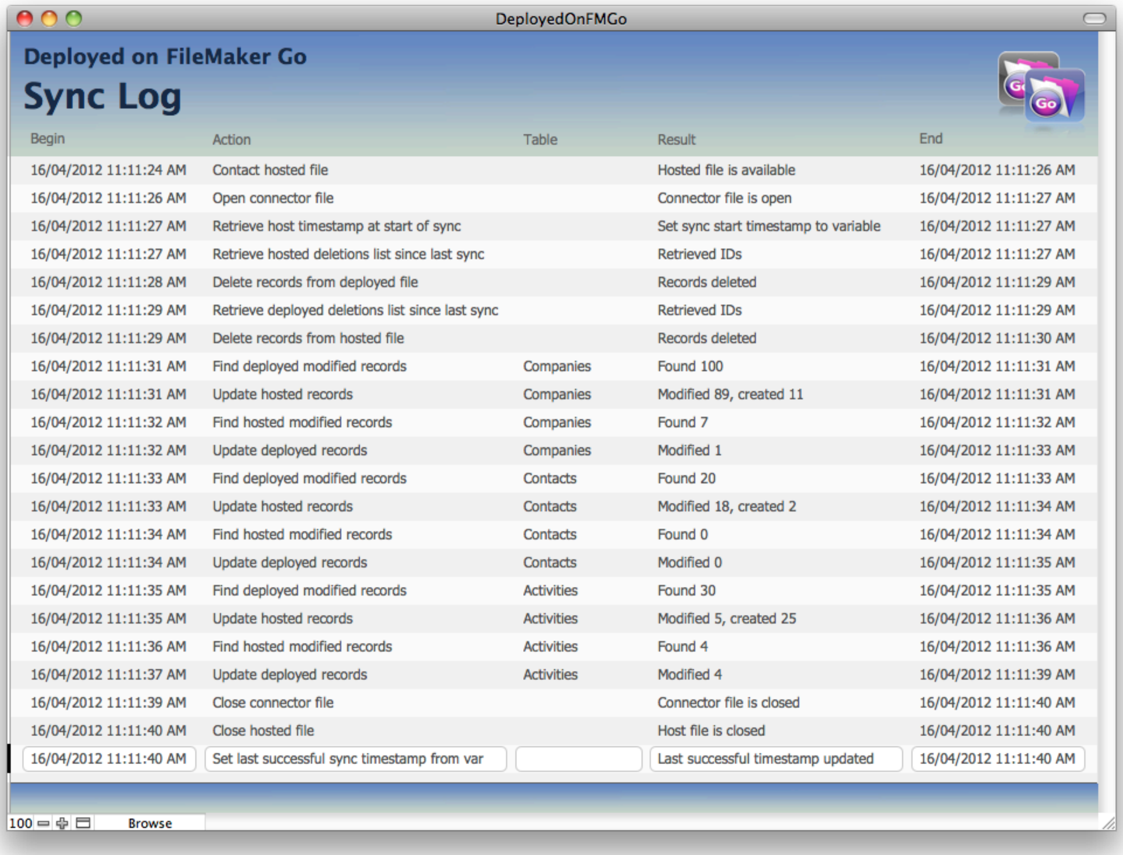
If the iOS device…	The hosted record will be…
Loses network connection	Not modified, locked*
Closes the file or window (user action)	Modified, not locked

Hibernates FileMaker Go	Not modified, locked*

*The locked record will be released when FileMaker Server determines that the user is no longer a client of the hosted file.

Keeping Track of Sync Status

To diagnose potential problems in the sync process, it may be useful to use a sync log to track which parts of the sync process have been completed. For instance, the following log displays a list of steps in a sync process:


DeployedOnFMGo

Deployed on FileMaker Go

Sync Log

Begin	Action	Table	Result	End
16/04/2012 11:11:24 AM	Contact hosted file		Hosted file is available	16/04/2012 11:11:26 AM
16/04/2012 11:11:26 AM	Open connector file		Connector file is open	16/04/2012 11:11:27 AM
16/04/2012 11:11:27 AM	Retrieve host timestamp at start of sync		Set sync start timestamp to variable	16/04/2012 11:11:27 AM
16/04/2012 11:11:27 AM	Retrieve hosted deletions list since last sync		Retrieved IDs	16/04/2012 11:11:27 AM
16/04/2012 11:11:28 AM	Delete records from deployed file		Records deleted	16/04/2012 11:11:29 AM
16/04/2012 11:11:29 AM	Retrieve deployed deletions list since last sync		Retrieved IDs	16/04/2012 11:11:29 AM
16/04/2012 11:11:29 AM	Delete records from hosted file		Records deleted	16/04/2012 11:11:30 AM
16/04/2012 11:11:31 AM	Find deployed modified records	Companies	Found 100	16/04/2012 11:11:31 AM
16/04/2012 11:11:31 AM	Update hosted records	Companies	Modified 89, created 11	16/04/2012 11:11:31 AM
16/04/2012 11:11:32 AM	Find hosted modified records	Companies	Found 7	16/04/2012 11:11:32 AM
16/04/2012 11:11:32 AM	Update deployed records	Companies	Modified 1	16/04/2012 11:11:33 AM
16/04/2012 11:11:33 AM	Find deployed modified records	Contacts	Found 20	16/04/2012 11:11:33 AM
16/04/2012 11:11:33 AM	Update hosted records	Contacts	Modified 18, created 2	16/04/2012 11:11:34 AM
16/04/2012 11:11:34 AM	Find hosted modified records	Contacts	Found 0	16/04/2012 11:11:34 AM
16/04/2012 11:11:34 AM	Update deployed records	Contacts	Modified 0	16/04/2012 11:11:35 AM
16/04/2012 11:11:35 AM	Find deployed modified records	Activities	Found 30	16/04/2012 11:11:35 AM
16/04/2012 11:11:35 AM	Update hosted records	Activities	Modified 5, created 25	16/04/2012 11:11:36 AM
16/04/2012 11:11:36 AM	Find hosted modified records	Activities	Found 4	16/04/2012 11:11:36 AM
16/04/2012 11:11:37 AM	Update deployed records	Activities	Modified 4	16/04/2012 11:11:39 AM
16/04/2012 11:11:39 AM	Close connector file		Connector file is closed	16/04/2012 11:11:40 AM
16/04/2012 11:11:40 AM	Close hosted file		Host file is closed	16/04/2012 11:11:40 AM
16/04/2012 11:11:40 AM	Set last successful sync timestamp from var		Last successful timestamp updated	16/04/2012 11:11:40 AM

100 Browse

Figure 4: A sync log with records for each step.

Identifying Records for Sync

In order for the sync script to identify which records to include, and to indicate which records have been synced at a specific time, each record requires a field that tracks when modifications have been made. This field, called zModStamp in this paper, should be automatically updated to the current timestamp during all modifications except those made during sync. During sync, the field should be updated to reflect the modification timestamp value of the record from which changes are being copied.

To manage this behaviour, use an auto-enter calculation (replaces existing value) that is dependent on two other fields: A globally stored timestamp field (in a System table in your solution) called gModOverride and an auto-enter modification timestamp called ModTrigger. The auto-enter calculation in zModStamp may use a formula such as:

```
Let(x = System::gModOverride; If(ModTrigger and x; x; ModTrigger)
```

The sync script can identify records for sync by comparing their zModStamp value to the timestamp of the last successful sync.

> Note: The fields above are named using a simple naming convention that could be called (not simply) "Hungarian notation, camel case, no spaces." Hungarian notation is a way to indicate the function of an object via its name. In the naming convention used here, letters are prefixed to some field names, such as "c" to calculation fields, "z" to housekeeping fields like timestamps, "g" to global fields, etc. Data fields such as text and number would normally not have a prefix. Camel case applies capitalization to the first letter of words in a field name, which improves readability of names with no spaces.

An alternative for identifying records for sync is to use a log of modifications and deletions. The sync script can use this log to identify records to be modified or deleted by comparing the record IDs and timestamps in the log with record IDs and timestamps in the tables to be addressed.

Deletions can be addressed in the same section of the script that handles modifications, by comparing the creation timestamp to the last sync timestamp. If the creation timestamp precedes the last sync timestamp and the record exists only in the deployed file or only in the hosted file, it was deleted in one system and should therefore be deleted in the other. If the creation timestamp is later than the last sync timestamp and the record exists in only one system, it is new and should be added to the other system.

Business Rules

As mentioned in "Processing Records" above, business rules will determine which modifications (when the same record has been modified by more than one user) take precedence over others. To establish your business rules, start with asking "When there is a conflict, who wins?" The answer may be straightforward, such as, "Remote users always win" or "The hosted file always wins." The answer could be more nuanced, such as "The record with the most recent modification timestamp wins" or "The record with the most recent timestamp wins but not if an administrator made an earlier change."

The following table lists a few business rules to be considered, with the corresponding record processing action.

If remote users…	Their sync process should…
Need records from the host only	Download records
Create new records	Upload new records
Modify records and take precedence over modifications made on the host	Upload modified records, overwriting hosted records
Modify records and do not take precedence over modifications made on the host	Modify hosted records during sync only if the record has not been modified on the host, and modify deployed records based on changes in hosted records

Delete records	Delete records with creation timestamps that are earlier than the last sync timestamp, where the records exist in one system but not the other

You may want to consider other factors to establish which changes your sync script will make, such as which users takes precedence among multiple remote users, the time of day or day of the week a change was made, the device on which the change was made (server or iOS), the group the user belongs to (such as managers or administrators), the location of remote users, and the distance away a remote user is. There are many factors not covered by this list. The point is to understand the applicable business rules and to apply their logic in the part of your script that modifies records.

There may be some cases that should be reviewed and reconciled manually by a person, and this too will reflect business rules. As each record is addressed during the sync process, any conflicts that cannot be resolved by the script should be posted to a list of exceptions. The list should include the unique record ID and field values that will help the reviewer decide which change to accept. A mechanism for review is beyond the scope of this document, but if you do create a review process, it will be useful to include portals, navigation and/or scripts attached to buttons to help the reviewer make changes efficiently.

Order of Record Creation/Modification

Another dimension of business rules has to do with record creation order. For instance, if your solution has tables for companies, invoices, line items and payments, you might also have a user who runs accounts payable reports. If that user pulls the report while a sync process is running, and records are updated in the payments table before the line items or invoices table, the accounts payable report may be inaccurate. The same problem may occur if records in customers, orders and line items are modified before payments.

To minimize this impact, you might choose to modify related records, if any, via portals or relationships before committing changes to the “parent” record. This ensures that the integrity of a set of related data is committed simultaneously as a single transaction, rather than modifying all records in a single table at one time and committing after each record has been modified. Making modifications to related records from a single context before committing saves the changes in cache until the commit; this method has the added benefit of not making modifications to the parent or any of the related records if the script is interrupted before the commit.

Also consider when the sync script should be allowed to run, such as outside the timeframe when reports are being pulled, and in which order table data should be updated.

Exchanging Data

If your business rules require that data from remote users be transformed or vetted before being integrated into the “live” system, it is recommended that the sync script in the deployed system only exchange records. Separate processes should subsequently process the received data in the hosted file and/or the deployed file. This ensures that the period of time when the sync process requires access to the host is minimized, allowing data processing to continue after the hosted file is closed.

In this case, you will need to create intermediary tables in one or both files. These tables may reside in the hosted and/or deployed files themselves, or in separate files dedicated to the purpose. The tables should be exact matches to existing, live tables, with the exception that auto-enter functions should be disabled. This is so that IDs, timestamps, etc. will not be modified when data are moved into these tables.

Record Creation

To move records into the intermediary tables rather than making modifications to live records during the sync, use Import script steps or loop through found sets of records and use Set Field script steps (via relationships for which the “Allow creation of records in this table via this relationship” option has been enabled) to create and then populate new records. See the section below called “Record Creation and Modification” for an explanation of creating records via a “pivot” relationship on the Relationships Graph.

Alternative Exchange Methods

Aside from importing or using relationships to move records from one table to another, records can be exported and emailed to another location and then imported.

Record data may also be passed as parameters in a Perform Script step or, using the Open URL script step, as parameters included in a string formatted in the FMP URL protocol. In either case, data may be passed back to the calling script in the script result. Calling a script using either method will cause the hosted file to be opened on the local device.

In the FMP URL protocol, non-alphabetical and non-numeric characters must be escaped using URL-encoding. Note that the inclusion of container data is not supported via the URL protocol. Moreover, container data cannot be exported to a non-FileMaker file, so the FileMaker export format is required for exchanging record data where container fields are included.

An additional option is to use a php-based system to exchange record data. Using web pages configured to work with the hosted file, the deployed or connector file can send URL requests to the pages to call scripts that add, edit, delete and retrieve records in the hosted file. By extension, the deployed or connector file may be configured to work via php with an SQL system to provide comparable functionality, or may utilize ODBC/JDBC technology to exchange data. Detailed discussion of these options is beyond the scope of this paper.

Going “Live” With Exchanged Records

Once records have been moved into intermediary tables, the sync script on the iOS device should close the connector file and the hosted file. The hosted file now has a new batch of records to process and integrate into the live system. Records may be processed by a script that is scheduled in FileMaker Server or that is run in a robot (a purpose-specific file running in a dedicated FileMaker client deployment). In either case, the process will operate continuously or at scheduled intervals to process queued data as appropriate.

The processor script may be configured to look for new records in the intermediary table(s) into which the sync process has transferred data. When the script detects an intermediary record awaiting processing, it should apply the relevant business rules to reconcile the intermediary data with the content of the corresponding record in the host.

Similarly, the deployed file may need to process intermediary records after the exchange of data with the hosted file is completed.

The processing of records requires the same logic as described earlier to determine which data should be overwritten during sync—the records that “win” are determined by business rules such as whether the record was modified in a deployed file or the host, whether the user has a particular role, etc. As noted previously, an override mechanism will be required to ensure that the modification timestamp in the synchronized data will match the modification timestamp of the source record (as opposed to reflecting the time when data exchange or post-processing occurred),

The specifics of comparing records, maintaining matching modification timestamps and primary keys and updating records in place are explored in the next section and in the "Processing Records" section of *The Sync Process Step by Step* later in this document.

Record Modification

Record data can be modified—either manually or via a scripted process—using an import that matches data in key fields or by field-by-field modification. The latter can occur through relationships or by directly navigating to the record that is to be updated on a layout based on the relevant table.

An import can be configured to add new records, update matching records or update existing records. While this is a direct and efficient method of updating records in one direction, it does not readily provide for records to be updated selectively based on business rules. Moreover, an import procedure will be required to update all fields that may be implicated whether or not they have changed, and the resulting impacts (arising from indexing and other secondary processes) may result in non-optimal sync performance.

An approach that offers greater control and may also be more efficient is to loop through each record, compare the value in each field, and modify only those fields that are different. This approach is described in more detail in the "Processing Records" portion of the section *The Sync Process Step By Step*.

Using Relationships to Compare and Modify Records

An efficient way to compare two records is via a relationship. This requires a table occurrence (also called a TO) based on the current context, meaning the table attached to the layout the sync script is on when the comparison begins, and a second TO based on the table to be examined, as shown in Figure 5 below. If the current context is not one of the tables to be compared, a third TO is required for the current context, as shown in Figure 6.

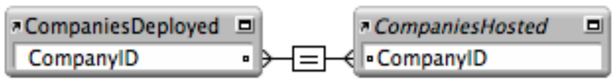


Figure 5: Table occurrences (TOs) for the Companies tables in both the deployed file and hosted file, related on the Company ID.

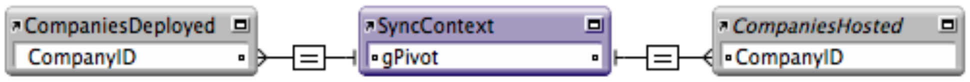


Figure 6: TOs in a relationship among three tables, with the center TO being the context of the sync script.

Generating Unique Primary Keys

The relationships above are based on primary keys in the tables being compared. Primary keys for each record must be unique, so that each record maintains its own identity and relationships with other records. For instance, a company may have related contacts, which may in turn have related activities. The primary key for the company ties the related contact records to that company, and the primary key for the contact ties the related activities records to that contact.

The goal of synchronization is to have a matched set of records in the deployed file and the hosted file. This includes the primary keys. In other words, while primary keys should be unique to each record, records and their relationships in deployed systems should reflect the records and their relationships in the hosted system exactly.

Generating unique primary keys using FileMaker's auto-enter serial number option may be sufficient when the requirement of the sync is to move new records from one source only, such as from the host to deployed files. However, when records may be created in both the hosted system and one or more deployed files, a method for avoiding primary key conflicts between records created in different copies of the same system will be required.

FileMaker's Get(UUID) function provides a robust alternative to the auto-enter serial number option. It generates a unique 32-character value on record creation that may be used as the primary key. For solutions created using FileMaker 12, this method of generating unique keys is recommended.

Using Timestamp Fields

One of FileMaker's field types is the timestamp, which combines a date and time and may be displayed in a format such as "4/18/2012 2:49:10 PM." Timestamp fields can be configured to automatically enter the date and time that a record is created or modified, or may be updated by a user or a script. In order to keep track of when records were modified be sure to include a field set to auto-enter the timestamp on modification. To facilitate synchronization, you should create an override mechanism to ensure that the modification timestamp in a synchronized record matches that of the source record

In addition to timestamps on individual records, the deployed file should have a single-record table in which to store the most recent successful sync timestamp. Records in the hosted file with a modification timestamp later than this timestamp should be included in the sync. (For efficiency, this single-record table may perform other functions, such as storing graphics, utility fields and other solution-wide data.) If you are confident that the deployed file will always be single-user, a global field can be used to store the last sync timestamp.

For users working in different time zones, be sure to adjust for the time difference before utilizing timestamps for sync. For instance, for users in a different time zone from the host, you will need to establish the time difference between the user's location and the host and add or subtract it from modification timestamp values in the deployed file before comparing them to timestamps in corresponding records in the hosted file. Where users move freely between time zones, the deployed file should capture a location value with each modification so that the appropriate time zone adjustment can be calculated.

Some options for establishing the time difference are to:

- store the relevant user's time zone in each copy of the deployed file
- prompt the user to enter and update a time zone value in a user preferences table in the deployed file
- "scrape" a web page that returns the current time of the host and parse the time value from the html source
- use the Location () or LocationValues () functions, which return the latitude and longitude of the device, and reference a stored table (or online source) correlating latitude and longitude with time zone.

Transactional Integrity

Because of the possibility of losing network connection, it is important that the sync scripts complete all changes to a given record or group of records before commit. Modifications should be done in a way that locks the record for the shortest possible time and makes relevant changes while the record remains locked. This will ensure that the complete set of changes is transmitted in a single action, providing transactional integrity. If the connection is lost during transfer of a record, the record will not have been committed and all the changes to it will be lost. Meanwhile, records that cannot be modified because they are locked should be added to the list of exceptions mentioned previously, and reported to the user at the appropriate time.

FileMaker provides record locking so that changes made by one user do not conflict with changes being made simultaneously by another user. Once the first user has committed the record, another user may then modify the record. The sync script may be programmed to attempt to modify locked records a number of times before adding the record to its list of exceptions.

To modify unlocked records, the script should be configured to lock the record, make changes and commit the record (thus releasing the lock) as efficiently as possible. This may entail comparing each field to its counterpart in the corresponding record, and changing only those fields that are different. Another option is to "pre-load" a variable with only the field names and values that should be changed and then use the values in the variable to change only those fields listed. A third option is to import one or more records using the Update Matching Records option.

No matter which method you use for record modification, the override mechanism should be used to ensure that the modification timestamp fields match, and reflect the actual time the change was originally entered rather than the current time at the point when sync occurred.

This method establishes the criteria described in "Identifying Records for Sync" above—records in either file with a modification timestamp equal to or greater than the timestamp of the last sync will be included in the current sync.

Note: In most cases it will be desirable to accept record changes as a whole and to thereby preserve transactional integrity. However, where business rules allow discrete changes to individual field values, you can provide a mechanism to track the modification timestamp for one or more individual fields in each record. The same modification-in-place method described below can be applied to synchronization of field-level modifications.

Opening and Closing Hosted Files

To avoid inconveniencing users, the sync process should open the hosted file when needed and close it when it is not required to be open, keeping the period of time that the hosted file is open to as short a time as possible. If a hosted file is not closed and the network connection is lost or interrupted, FileMaker Go will pause while attempting to reconnect and may prompt the user to re-authenticate when the network connection is re-established. It is desirable to keep the potential for connection-related delays or prompts to a minimum.

As mentioned at the start of this section, FileMaker Go will allow hosted files to be closed only if no other open files have table occurrences referencing tables in the hosted file. The following chart lays out the methods for closing hosted files on an iOS device:

If the deployed file…	The hosted file will open if…	The hosted file will close if…

has a file reference to the host	Open File [hosted file] script step or the Perform Script [script in the hosted file] script step is called	Close File [hosted file] script step or Perform Script [script in the hosted file that closes the file] script step is called
has a TO based on a table in the hosted file	Above action or any script steps that reference fields in the hosted file (such as Set Field []) are called	Current file and any other files with table occurrences based on the hosted file are first closed
has a layout based on a TO that points to a table in the hosted file	Above actions or user or script navigates to the layout	Current file and any other files with table occurrences based on the hosted file are first closed
contains a script or calculation that references a field in a TO based on a table in the hosted file, or has any layout containing a field from the hosted file	The relevant script runs, the relevant calculation is evaluated and/or the relevant layout is displayed	Current file and any other files with table occurrences based on the hosted file are first closed

The implication for closing a hosted file in a sync scenario is that any files with table occurrences based on tables in the hosted file must be closed before the hosted file can be closed. If the deployed file has any of these table occurrences, it must be closed before the hosted file is closed. This can be avoided by having only a file reference (but no table occurrences) pointing to the hosted file in the deployed file, and using a separate, connector file also opened on the deployed device to handle relationships between table occurrences based on tables in the deployed file and the hosted file. With only a file reference to this connector file in the deployed file, the deployed file can close the connector file after sync is complete and can then close the hosted file.

To open and close the hosted file as needed, opening, syncing and closing files should be performed in this order:

1. The deployed file opens the hosted file. If this is successful, the sync process continues.
2. The deployed file opens the connector file, which has table occurrences to both the hosted file and the deployed file.
3. The deployed file calls a script in the connector file to sync records.
4. The deployed file closes the connector file.
5. The deployed file closes the hosted file.

The Sync Process in Plain Language

In this section, the sync process is laid out as an outline, with the goal of assisting your planning and production process. Throughout the process, remember to log actions, monitor for errors, halt the process if necessary, and alert the user when appropriate. The section following this one sets out the essential components of a sync process both conceptually and with specific examples of FileMaker script steps.

Part 1—In the Deployed FileMaker Go File

1. The user launches the sync process by tapping a button in the deployed file.
2. The script in the deployed file checks to see if the hosted file is open by calling a 'handshake' script in the hosted file.
    a. After confirmation that the hosted file is available, the script in the deployed file exports and opens the connector file, then waits for the connector file to run its portion of the sync process.
    b. If the hosted file is not available, the script notifies the user and halts.

Part 2—In the Connector File

1. The sync script runs when called by the deployed file.
2. The script in the connector file collects a list of record IDs or performs a Find for modified records in each table in the *deployed* file.
3. Depending on whether records will be processed by this script or not:
    a. If the connector file is configured to exchange records only, it pushes record data into intermediary tables in the *hosted* file. It may also collect and upload a list of the primary keys for records that have been deleted in the deployed file since the last sync.
    b. If the connector file is configured to process record sync, it loops through the record IDs or records and performs the required modifications on related *hosted* records, including creating new records and deleting records deleted since the last sync.
4. The script in the connector file repeats steps 2 and 3 (above) with the direction of synchronization reversed (i.e. with the host file as the data source and the deployed file as the data recipient).
5. The script concludes by updating the hosted sync log record and creating a log record in the deployed file.

Part 3—In the Deployed FileMaker Go File

1. The script closes the connector file.
2. The script closes the hosted file. If records are being processed after data exchange, the script now processes deployed records, looping through records in intermediary tables, comparing timestamps, and updating, creating or deleting deployed records according to business rules.
3. The script updates the deployed file's sync log and, if the sync completes successfully, writes the sync start timestamp into the System::gSyncTimeStamp field in the deployed file.
4. The script notifies the user that the sync process on the device is complete (for instance, by displaying a custom dialog or updating a field on a layout).

Part 4—In the File Hosted by FileMaker Server or a Stand-Alone FileMaker File (if necessary)

1. A scheduled script checks for new records in intermediary tables. The script may be scheduled to run either in FileMaker Server or in a robot, a separate FileMaker client file configured to run scripts on a regular basis.
2. The script performs its updates, looping through records in intermediary tables, comparing timestamps, and updating, creating or deleting live, hosted records according to business rules.
3. The script updates the host file's sync log table with the sync status and completion timestamp.

The Sync Process Step by Step

With the sync strategy planned from the iOS device's perspective, it is time to work out the specifics of the process. Because sync requirements vary widely, no single approach will apply to all situations. With that in mind, this section describes the main components of common sync scenarios and includes options for variations. For a summary of the components, see the previous section, *The Sync Process in Plain Language.*

> **Note:** The sample script steps in this section are intermediate to advanced in complexity. Some advanced techniques include the use of variables, the Open URL script step and the ExecuteSQL( ) calculation function. If you are not familiar with these or other techniques in the following examples, you are encouraged to refer to training materials and other resources for guidance. Links to downloads and materials that reference the techniques described here can be found in the Resources section.

The sync process developed for this section is based on these assumptions:

- The files to be synchronized are a deployed file and a hosted file, and an additional file is used to manage the sync tasks. For the purposes of these examples, the three files are called HostedOnServer.fmp12, DeployedOnGo.fmp12, and SyncConnector.fmp12.
- Users of the deployed file modify, create and delete records in the deployed file.
- Users of the hosted file modify, create and delete records in the hosted file.
- The sync performs record modification, creation and deletion.
- The sync may work with live data or may move records into intermediate tables first.
- Tables and fields in the deployed file either match or are a subset of those in the hosted file, including using the same table and field names as those in the hosted file.

The issues described in the *Developing Your Sync Strategy* section above are addressed as follows:

- Planning from the iOS Device's Perspective
    - The process is launched from within the deployed file.
    - The initial sync script includes Allow User Abort [Off] and Set Error Capture [On] (sub-scripts inherit these settings). See *Developing Your Sync Strategy* above for a detailed discussion of the use of these options in sync scripting,
    - All scripts involved in the sync process run with full access privileges.
- Network Connection
    - Establishing Server Availability
        - The deployed file establishes whether the hosted file is available and either aborts the process if the hosted file is not available or allows the process to continue if the hosted file is available to sync.
        - If the Hosted File is Unavailable
            - The deployed file halts the sync script if the server is unavailable.
        - Consequences of Losing Network Connection
            - The sync script includes tests at key points to determine whether the hosted file is still available. If not, the sync script creates a log record and halts.
        - Consequences of Hibernating FileMaker Go
            - If FileMaker Go is re-activated, it will try to re-establish its connection with the hosted file and resume any running scripts. Again, if the hosted file is unavailable, the sync script creates a log record and halts.
        - Reconnecting and Re-Authenticating
            - The fmreauthenticate extended privilege is enabled and set to 60 minutes to allow time for remote users to re-establish a network

connection. In your own sync scenario, this amount of time should be set according to what you determine to be a reasonable interval in light of user work requirements and network availability.

- Keeping Track of Sync Status
  - Both the deployed file and the hosted file have “sync log” tables. The sync scripts create log records for each action in the process.
- Business Rules
  - To handle deleted records, the script compares the primary key in each record to a matching primary key in the corresponding table. If there is no matching record and the record creation timestamp is earlier than the last successful sync timestamp, the record will be deleted.
  - Other business rules in this sample include the “most recent modification wins,” regardless of which file the change was made in.
- Identifying Records to be Synced
  - The sync script performs a Find or uses the ExecuteSQL( ) function to collect record IDs in both files, isolating records that have been created, modified or deleted in either file since the last sync.
- Order of Record Creation/Modification
  - This example processes all records one table at a time. In a time-sensitive scenario such as described earlier, you might choose to process related records starting with a parent record.
- Exchanging Data
  - The deployed file launches a local “connector” file to manage record exchange and modification via relationship. This is primarily so that the hosted file can be closed when data exchange or sync is complete. To address other options, alternative record creation and exchange methods are also discussed. The connector file is stored in a container field in either the hosted file or the deployed file and is deployed to the iOS device as part of the sync process. This eliminates the need for users to separately install or manage the connector file.
- Record Modification
  - Relationships
    - The connector file contains relationships to each of the tables in the deployed file and the hosted file.
  - Primary Keys
    - The deployed file and the hosted file both generate primary keys using the Get(UUID) function. Alternatively, unique primary keys may be generated using a customized calculation.
  - Timestamps
    - A modification timestamp is used in each table using a trigger and an override field to ensure that the original timestamp gathered at the point of user modification will be preserved during sync (rather than being overwritten by the current timestamp when the record is updated during the sync procedure). This technique works by allowing the original (source record) modification timestamp to be placed in an override field (a global field in a “System” table) and to take precedence over the current timestamp value in a calculation that returns the zModStamp value for the current record. A conventional auto-enter timestamp field is used as the trigger for the calculation that produces the modification timestamp. In addition, a global timestamp field in the System table in the deployed file is used to store the starting timestamp of the last sync that completed successfully.
  - Transactional Integrity
    - The sync script loops through the fields in each record, comparing the related fields in the matching record, to determine which fields should be modified. The

additional option of “pre-loading” a variable with fields to be modified is also discussed.

- Opening and Closing Hosted Files
    - Three files must be open for this sample sync process to work—the deployed file, the hosted file and a connector file deployed on the iOS device.
    - The deployed file opens the hosted file first, and then the connector file. If the hosted file is not available, the deployed file halts the script before opening the connector file.
    - When sync is complete, the deployed file closes the connector file and the hosted file. Because the hosted file can be closed only when there are no other files open with table occurrences to the hosted file, the connector file must be closed first. In return, the connector file cannot be closed if there are table occurrences based on its tables in the deployed file, so be sure that the deployed file does not have TOs to the connector file. Once the connector file is closed, the deployed file can close the hosted file.

The following sections include sample scripts, additional options and details expanding on and providing a basic implementation outline for a sync process based on the concepts above.

Contacting the Host

The first step in the sync process is for the deployed file to contact the host. The following script steps handle contacting the host, alerting the user if the host is not available for any reason and logging the result. To establish the availability of the hosted file, the script attempts to set the user’s ID from the hosted system (which in this example is the primary key in the users table) into a global field in the deployed file’s System table. If the action succeeds, the hosted file is available and the sync process can commence; if not, the script informs the user that network connection was not successful and creates a log entry documenting the attempt.

This script is designed to run in the deployed file, on a layout based on the deployed file’s System table.

```
Set Error Capture [On]
Allow User Abort [Off]
Set Field [System::gPivot; “”]
Set Field [SyncLog::Action; “Contact hosted file”]
Commit Records/Requests [Skip data entry validation; No dialog]
Set Variable [$token; Value: “Handshake_” & GetAsNumber(Get(CurrentTimestamp))]
#The SyncHandshake script is in the HostedOnServer file and is described below.
Perform Script [“SyncHandshake” from file “HostedOnServer”; Parameter: $token]
If [Let(
        $error =
        Case(
        Get(LastError) = 100; "The hosted file is missing";
        Get(LastError) = 802; "Unable to open the hosted file";
        Get(ScriptResult) ≠ $token; "The hosted file is currently unavailable"
        );
        not IsEmpty($error)
        )]
   Show Custom Dialog [Title: "Connection Error"; Message: $error & “.” Default Button:
   “OK”, Commit: “No”]
   Set Field [SyncLog::Result; $error]
   Halt Script
Else
   Set Field [SyncLog::Result; “Hosted file handshake successful”]
End If
```

```
Commit Records/Requests [Skip data entry validation; No dialog]
```

In the above steps, the script attempts to perform a script in the HostedOnServer file. If the result of Get(LastError) is a non-zero value or if the script result does not equal the supplied token passed as a parameter to the script in the hosted file, a custom dialog will inform the user that the sync will not continue, and the script creates a record of the failure in the sync log. The relationship to the sync log should be set to allow creation of records; see the section titled “Record Creation and Modification” above for more detail on this powerful feature of relationships, which is used extensively in the sample sync scripts.

Following is the suggested content of the “SyncHandshake” script in the HostedOnServer file:

```
Exit Script [Result: Get(ScriptParameter)]
```

A note about writing scripts: It is possible to write very compact scripts by using calculations in variables to handle variations in conditions. In these sample scripts, variables are typically named with one or two characters, such as $e for error and $ts for timestamp. Variables are used for many purposes such as storing timestamps and lists of record IDs and calculating field names for the Set Field By Name script step. Variables are set and modified in Set Variable steps as well as in the options for Exit Loop If[ ], If[ ], Else If[ ], and Set Field[ ] steps.

Deploying a Connector File

If you are using a connector file, it may be deployed on the iOS device or it may be stored in a container in the deployed or hosted files, exported to the iOS device and opened. Assuming that the connector file is stored in a field called gContainer in a System table in the deployed file, the following script steps in the deployed file will export and open the connector file:

```
Set Field [System::gPivot; “”]
Set Field [SyncLog::Action; “Export and open connector file”]
Commit Records/Requests [Skip data entry validation; No dialog]
Set Variable [$path; Value: "file:" & Get(DocumentsPath) & "SyncConnector.fmp12"]
Export Field Contents [System::gContainer; “$path” ]
Set Variable [$e1; Value: Get(LastError)]
If [$e1 = 0]
    Open URL ["FMP://~/SyncConnector.fmp12?script=Run Sync"; No dialog]
    Set Variable [$e2; Value: Get(LastError)]
End If
If [$e1 or $e2]
    Set Field [SyncLog::Result; Case($e1; “Export failed with error ” & $e1 & “.”; Connector file
          open command failed with error “ & $e2 & “.”)]
    Show Custom Dialog ["System Error"; Case($e1;“File export failure [error = “ & $e1 & “]”;
          “Connector file could not be opened [error = “ & $e2 & “]”)]
    Halt Script
Else
    Set Field [SyncLog::Result; “Connector file is open”]
End If
Commit Records/Requests [Skip data entry validation; No dialog]
```

The first steps clear the gPivot field that manages the relationship to the SyncLog table, then use the “allow-create…” relationship to the SyncLog TO to create a new log record to track the status of exporting and opening the connector file. The next step, Set Variable, creates a variable with the path to FileMaker Go’s own documents folder. The Export Field Contents step exports the file from a global container field in the System table of the deployed file. After setting a variable with the result (to be used for error handling, logging and reporting in both the following If[ ] sections), the

Open URL step uses the FMP URL protocol to send a call to the newly exported file to run a script called OnFileOpen. Calling a script this way effectively opens the file on FileMaker Go.

After the Open URL step, the script stores the LastError result from the Open URL step, displays an error dialog if necessary, and sets the result of the log record.

> **Tip:** To be sure that the sync process is running with the latest version of the connector file, you could add steps to the OnFileOpen script in the connector file to replace itself if a newer version is available from a container field in the host, before proceeding with the sync.

The connector file is designed to be the center of data exchange (and record processing if both functions are performed at the same time). Its purpose is to identify records for sync, compare record data, exchange and/or modify records, and create log records. The two tasks it does not perform are opening and closing the hosted file, as these are performed directly by the script in the deployed file that initiates the process.

To perform its assigned tasks, the connector file must have:

- file references to both the deployed file and the hosted file
- table occurrences that connect corresponding tables in both the deployed file and hosted file
- a table with appropriate fields, including a globally stored text or number field with which to create relationships to other TOs and a globally stored timestamp field
- relationships between the TO for the table in the connector file and each of the other TOs (see Figure 7 below)
- a layout based on the TO for the table in the connector file
- layouts based on the TOs of the deployed file and hosted file if the sync script uses any Go to Layout or Go to Related Records steps
- a script called by the deployed file to identify eligible records; to import, export or modify records according to business rules; and to create log records

The following figure shows how the Relationships Graph can be arranged to meet the above criteria:

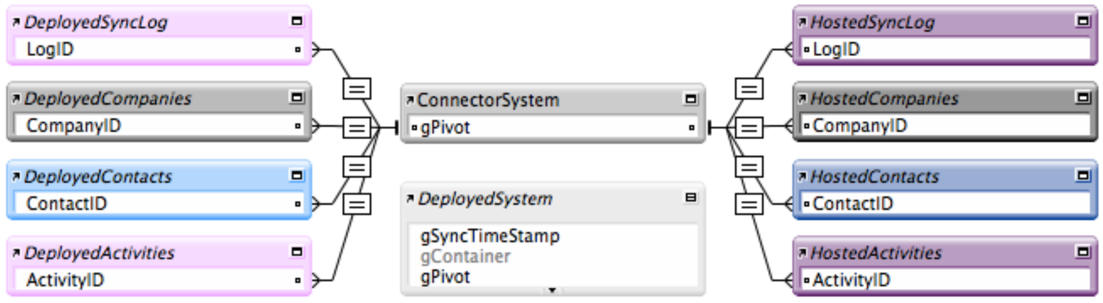


Figure 7: The Relationships Graph in the connector file

In the ConnectorSystem TO, gPivot is a text field with global storage. It functions as a key field to create “pivot” connections between the deployed tables and hosted tables. A primary key can be set into the gPivot field to create a valid relationship to one or more tables at the same time.

The DeployedSystem TO is on the graph so that the sync script can work with the field that stores the last successful sync timestamp—in this case, the global field in the deployed file called gSyncTimeStamp. Global storage of the gSyncTimeStamp field allows the sync start timestamp for the last successful sync to be available to the entire solution without a relationship.

The Run Sync script in the SyncConnector.fmp12 file should begin by navigating to an appropriate layout and setting variables with appropriate timestamp values for reference throughout the sync process:

```
Go to Layout [“SyncCentral” (“ConnectorSystem”)]
Set Field [ConnectorSystem::gPivot; “”]
Set Field [DeployedSyncLog::Action; “Retrieve timestamp values for sync run”]
Set Variable [$SyncTimeStamp; Value: DeployedSystem::gSyncTimeStamp]
Set Variable [$CurrentSyncTimeStamp; Value: Get(CurrentTimeStamp)]
Set Field [DeployedSyncLog::Result; “Sync start timestamp stored as a variable”]
Commit Records/Requests [Skip data entry validation; No dialog]
Set Field [ConnectorSystem::gPivot; “”]
Set Field [HostedSyncLog::Action; "Sync commenced by " & DeployedSystem::gUserID & " at
    " & $CurrentSyncTimeStamp]
Set Variable [$HostSyncLogID; Value: ConnectorSystem::gPivot]
Commit Records/Requests [Skip data entry validation; No dialog]
```

The DeployedSystem::gSyncTimeStamp field, mentioned in the section on timestamps above, is a globally stored field that holds the last successful sync timestamp. It is set into a variable to be used later in the sync script and is re-set to the value in $CurrentSyncTimeStamp upon successful completion of the sync. If the sync does not complete successfully, the value in DeployedSystem::gSyncTimeStamp will not be updated, and the next sync attempt will work with records updated since the previous successful sync, rather than records that may have been deleted or modified since the unsuccessful sync.

In this sequence of steps, variables are set with the last successful sync timestamp and the current timestamp. Setting a variable with the last successful sync timestamp means that the value must be pulled from a field only once, and can be referenced from a variable for the duration of the script. The $CurrentSyncTimeStamp variable stores the timestamp value immediately prior to the creation of a log record marking the beginning of the actual sync. At the end of the sync script, if the sync completes successfully, this value will be written into the DeployedSystem::gSyncTimeStamp field, representing the starting point for the next round of deletions and modifications.

Log records are created by first clearing the gPivot field and then setting a value into a field in the log table (the relationship to the log table has the “Allow creation of records in this table via this relationship” option enabled). A log record is also created in the hosted sync log, and the ID of that log record is stored in a variable so that it can be used at the end of the script to record the result of the sync (i.e. whether it was successfully completed) in the hosted log table.

Identifying Records for Sync in the Deployed File

All modified records in both the deployed file and the hosted file must be addressed for a complete sync. This can be achieved by comparing the zModStamp field in each record with the $SyncTimeStamp value. Records in either file that have a modification timestamp that is later than the start of the last sync are candidates for inclusion in the current sync. In cases where each user will require only a subset of records from the host, sync candidate records on the host will need to be selected according to the criteria that define the user’s subset, in addition to the modification timestamp criterion.

In applying the criteria that make sense in your solution, you may choose to work from the context of the tables in each file (the host file and the deployed file) in turn, comparing the records in each to the other. Or you may prefer to work from a single vantage point, addressing the records in both files via relationships from one context.

The three examples below offer different methods for identifying records for sync: Performing a Find, collecting record IDs in a variable using specified variable and table names, and collect the record IDs in variables using indirection (in which the variables and tables are determined when the script runs).

Isolating Records by Performing a Find

The following steps provide an example of a method you may use to isolate records in a single table in the deployed file based on the most recent successful sync timestamp. Steps in italics may be replaced by the steps in the following two methods:

```
Freeze Window
Set Field [ConnectorSystem::gPivot; “”]
Set Field [DeployedSyncLog::Action; “Find modified records in the deployed file”]
Commit Records/Requests [Skip data entry validation; No dialog]
```
Go to Layout [“DeployedCompanies” (DeployedCompanies)]
Perform Find [Restore; DeployedCompanies::zModStamp: “> $SyncTimeStamp”]
Set Field [System::gPivot; “”]
Set Field [DeployedSyncLog::Result; “Found ” & Get(FoundCount) & “records”]
Set Field [DeployedSyncLog::Table; Get(LayoutTableName)]
Commit Records/Requests [Skip data entry validation; No dialog]

Alternatively, you could use a Go to Related Records step in place of the Go to Layout and Perform Find steps, using a relationship that filters records based on a “>” relationship between a global timestamp field and the zModStamp field (prior to the Go to Related Records step, your script should set the value of the $SyncTimeStamp variable into the global key field. Whichever method you choose, the steps to isolate candidate records for inclusion in the sync should be performed for each table to be synced.

Collecting Record IDs in a Variable

To work from a single context, rather than performing a Find in each table and looping through the records from each table’s context, your script could instead gather the IDs of records to be synchronized. First, the script should capture the key values of modified records. Once the script has gathered the primary key values of the records to be synchronized, there will be no need to use a Go to Related Records script step, as the script can address each record in turn by placing one ID at a time into the System::gPivot field in order to instantiate a relationship between the source table and the destination table. In this case, the source table would be DeployedCompanies and the corresponding table would be HostedCompanies.

An example of a script sequence that captures the IDs (which can be used in place of the italicized steps for each table above) is:

```
Set Variable[$CompanyIDs; Value: ExecuteSQL("Select CompanyID from
    DeployedCompanies where zModStamp > ?"; ""; ¶; $SyncTimestamp)
Set Field [DeployedSyncLog::Result; “Found ” & ValueCount($CompanyIDs) & “records”]
Set Field [DeployedSyncLog::Table; “Companies”]
Commit Records/Requests [Skip data entry validation; No dialog]
```

If you are using the method outlined in the above example, be sure to change the variable name and the value set into the log’s Table field for each table.

Collecting Record IDs in Variables Using Indirection

As an alternative to “hard coding” the steps to set a specific variable for the IDs from each table, you may prefer to use a loop to collect the IDs for each table and an additional calculated variable per loop to create a unique variable for each table. The following steps use variables to accumulate IDs for each table and to create the variable names themselves:

```
Set Variable [$refList; Value: "DeployedCompanies|CompanyID¶
    DeployedContacts|ContactID¶DeployedActivities|ActivityID" ]
Loop
    Exit Loop If [
        Let([
        v1 = Substitute(GetValue($tables; 1); "|"; ¶);
        $table = GetValue(v1; 1);
        $field = GetValue(v1; 2);
        $refList = RightValues($refList; ValueCount($refList) - 1)];
        IsEmpty($table)
    ) ]
    Set Field [ConnectorSystem::gPivot; “”]
    Set Field [DeployedSyncLog::Action; “Locate modified records in deployed table: ” &
        $table]
    Set Variable [$IDs; Value: Evaluate("ExecuteSQL(\"Select " & $field & " from " & $table &
        " where zModStamp < ?\"; \"\"; \¶; $SyncTimeStamp)")]
    Set Variable [$countVar; Value: ValueCount($IDs)
    Set Variable [$calculatedVar; Value: Evaluate(
        "Let($" & $table & "IDs = \"" & Substitute($IDs; ¶; "\¶") & "\"; \"\")")]
    Set Field [DeployedSyncLog::Result; “Located “ & $countVar & “ records”]
    Set Field [DeployedSyncLog::Table; $table]
    Commit Records/Requests [Skip data entry validation; No dialog]
End Loop
```

When this portion of the script is complete, you will have a set of variables named for each of the tables in the deployed file that have records eligible to be synchronized. In this example, the variables would be $DeployedCompaniesIDs, $DeployedContactsIDs and $DeployedActivitiesIDs. The values in these variables can be then used to address and process records individually via the relationship by placing each key value in turn into the gPivot field. For clarity in the Relationships Graph, table occurrences in this example have been named using the plural of the associated tables; however, naming them with the singular form may make it easier to read and use the calculated variables above.

Exchanging Data

If your sync scenario calls for record data to be exchanged first and processed at a later time, your script should only exchange data initially. You may choose simply to transfer all modified records between the file where they have been modified and the file to be synchronized, or to loop through eligible records and move only those that should “win” over their counterparts. The process of examining each record and either moving it to an intermediary table or making modifications during the sync script is described in “Processing Records” below.

To simply move all modified records, the following set of steps imports records from the deployed Companies table to the intermediate hosted table, then isolates modified records in the hosted Companies table and moves them into the intermediate deployed tables. After the Perform Find[ ] or Go To Related Records[ ] steps in the preceding examples:

```
Import Records [“SyncConnector.fmp12”]; [import from “DeployedCompanies”
    to “IntermediateHostedCompanies”; Add; Matching Names; Auto-enter Off]
```

```
Go to Layout [“HostedCompanies” (HostedCompanies”]
Perform Find [Restore; HostedCompanies::zModStamp: [≥ $SyncTimeStamp]]
Import Records [“SyncConnector.fmp12”]; [import from “HostedCompanies”
    to “IntermediateDeployedCompanies”; Add; Matching Names; Auto-enter Off]
```

Remember to include steps (as in previous examples) to create a log record and to set the Action, Result and Table fields into the log record as appropriate.

Note: If users are allowed to delete records in either system, it will be necessary to exchange a list of records in each table that were created before the last successful sync and do not exist in the corresponding table. See the section “Identifying Records to be Deleted” for a method to collect these primary keys. These lists may then be used by the “processing” scripts to delete records.

Alternate Record Creation Method

An alternative to importing records is to loop through either a found set or a list of record IDs and use the “pivot” relationship to create records and populate the fields. See the section below called “Record Creation and Modification” for an explanation of creating records via a pivot relationship. Be sure to commit the record after all fields have been populated and clear the gPivot field before setting the fields for the next record.

Alternate Exchange Methods

As mentioned above, the FMP URL protocol allows you to send a call to a hosted file to run a script. This provides an alternative way to move records to a hosted file. For example, an OpenURL script step in the form set out below calls a script named “CreateNewRecord” in a hosted file called “HostedOnServer,” and passes variables for the table name, the record ID and the company name, populated with data from fields:

```
Open URL [ "FMP://{ServerIPAddress}/HostedOnServer.fmp12?
    script=CreateNewRecord&$Table=Companies&$ID=" &
    DeployedCompanies::CompanyID & "&$Company=" &
    DeployedCompanies::Company]
    [ No dialog ]
```

Going “Live” With Exchanged Records

Also as mentioned above, the sync script should notify the hosted file or intermediary file when records have been exchanged and hosted records are ready to be processed. This may take the form of creating a log record in the hosted file:

```
Set Field [SyncConnector::gPivot; “”]
Set Field [HostedSyncLog::Action; “Record exchange complete (user ” &
System::gUserID & “)”]
Set Field [HostedSyncLog::TimeStampEnd; Get(CurrentHostTimeStamp)]
Commit Records/Requests [Skip data entry validation; No dialog]
```

For record exchange only, that is the end of the sync script in the connector file. Next, the sync script in the *deployed* file should close the connector file and the hosted file:

```
Set Field [System::gPivot; “”]
Set Field [SyncLog::Action; “Close connector file”]
If [Position(¶ & DatabaseNames & ¶; ¶ & "SyncConnector" & ¶; 1; 1)]
    Close File [“SyncConnector”]
End If
Set Field [SyncLog::Result; “Connector file closed”]
Commit Records/Requests [Skip data entry validation; No dialog]
```

```
Set Field [System::gPivot; “”]
Set Field [SyncLog::Action; “Close hosted file”]
If [Position(¶ & DatabaseNames & ¶; ¶ & "HostedOnServer" & ¶; 1; 1)]
    Close File [“HostedOnServer”]
End If
Set Field [SyncLog::Result; “Hosted file closed”]
Commit Records/Requests [Skip data entry validation; No dialog]
```

The If [ ] statements test for whether the SyncConnector and HostedOnServer files are open. While this test is not strictly necessary, it is good practice to perform an action on a file only if the file is available.

In the deployed file, the script should then process records in its own intermediary tables. This portion of the script, and the script that will process records on the host, should follow the same strategies as described in the next section.

Processing Records

Finally, the sync process has reached the point of updating records. This includes modifying existing records, deleting deleted records and creating new records. The sync script should loop through records, compare matching modification timestamps (if any), and employ business rules to determine which records are to be modified. The loop can be configured to step through the found set of records or to use the list of record IDs collected in the $IDs variable or variables as described in the “Identifying Records for Sync in the Deployed File” section above.

Processing Hosted Records in Place

Processing “in place” is a flexible and powerful method for working with records in multiple tables without having to change context (which is efficient, and allows you to control when record sets are committed in order to maintain the integrity of transactions spanning multiple records). This method requires that a TO for the SyncConnector’s System table be connected to a TO for each of the deployed and hosted tables, as shown in Figure 7. Any or all of the deployed and hosted TOs may be pointed at intermediary tables instead of the live tables, depending on whether you want to create records in intermediary tables or work with records in the live system. On the SyncConnector::System side, a global text field must be related to the auto-enter ID field in each of the deployed and hosted TOs, and “Allow creation of records in this table via this relationship” must be enabled on the deployed or hosted sides of each relationship.

Using the list of IDs in the $IDs variable or variables, the subsequent script steps establish a relationship to one record in the deployed file and the matching record in the hosted file, from the context of the SyncConnector’s System table. The script then compares the modification timestamp of the deployed record to the modification timestamp of the corresponding hosted record. If the deployed record has been modified more recently than the hosted record and the hosted record is not locked, the script modifies fields in the hosted record for which the values in the deployed file have been changed. If the hosted record is locked, the script moves the record ID to the bottom of the list, assigns it to a temporary “locked IDs” list, and then to an exceptions list if the record cannot be modified after three attempts.

In addition, if there is no matching hosted record, the following steps will either create a new record on the host via the “Allow creation of records in this table via this relationship” feature of the relationship (where the deployed record has a creation date since the last successful sync), or will delete the record in the deployed file if the creation timestamp is earlier than the last sync timestamp (indicating that it is not a new record created since the last sync and has therefore been deleted from the host since the last sync).

Starting with the Set Variable step that establishes $IDs from “Identifying Records for Sync in the Deployed File” above:

```
Set Variable[$IDs; Value: ExecuteSQL("Select CompanyID from DeployedCompanies where
    zModStamp > ?"; ""; ¶; $SyncTimestamp)]
Set Field [SyncConnector::gPivot; “”]
Set Field [DeployedSyncLog::Action; “Update hosted records”]
Set Variable [$SyncLogID; Value: SyncConnector::gPivot]
Commit Records/Requests [Skip data entry validation; No dialog]
Loop
    Exit Loop If [Let([
        $id = GetValue($IDs; 1);
        $IDs = RightValues($IDs; ValueCount($IDs) - 1)];
        IsEmpty($id)
        )]
    Set Field [SyncConnector::gPivot; $id]
    If [DeployedCompanies::zGenStamp > $SyncTimestamp or
    (HostedCompanies::zModStamp and DeployedCompanies::zModStamp >
    HostedCompanies::zModStamp)
        #If the current record was created after the last sync timestamp or there is a
        corresponding hosted record, set the appropriate fields (this will either create a new
        record or modify an existing one).
        Set Field [HostedSystem::gModOverride; DeployedCompanies::zModStamp]
        Set Field [HostedCompanies::Company; DeployedCompanies::Company]
        If [Get(LastError) = 301]
            Set Variable [$variables; Value: Let([
            $IDs = $IDs & If(PatternCount($LockedIDs; $id) < 3; Left(¶; not IsEmpty($IDs)) &
                $id);
            $LockedIDs = $LockedIDs & Left(¶; not IsEmpty($LockedIDs)) & $id;
            $ExceptionsList = $ExceptionsList & If(PatternCount($LockedIDs; $id) > 3; Left(¶;
                not IsEmpty($ExceptionsList)) & $id)];
            ""
            )]
        Else
            Set Variable [$modCount; Value: Let([
                    $modCount = If(HostedCompanies::Serial#; 1) + $modCount;
                    $createCount = If(IsEmpty(HostedCompanies::Serial#); 1) + $createCount];
                    "")]
            Set Field [HostedCompanies::Location; DeployedCompanies::Location]
            Commit Records/Requests [Skip data entry validation; No dialog]
            Set Field [HostedSystem::gModOverride; “”]
        End If
    Else If [DeployedCompanies::zGenStamp < $SyncTimeStamp and
    IsEmpty(HostedCompanies::CompanyID]
        #If the current record was created before the last sync and not modified since AND
        there is no corresponding hosted record, delete the deployed record.
        #Be sure named portals are on the current layout
        Go to Object [Object Name: CompaniesPortal]
        Go to Portal Row [Select; First]
        Set Field [DeployedCompanies::Company; DeployedCompanies::Company]
        If [Get(LastError) = 301]
            Set Variable [$variables; Value: Let([
                $IDs = $IDs & If(PatternCount($LockedIDs; $id) < 3; Left(¶; not IsEmpty($IDs)) &
                $id);
                $LockedIDs = $LockedIDs & Left(¶; not IsEmpty($LockedIDs)) & $id;
                $ExceptionsList = $ExceptionsList & If(PatternCount($LockedIDs; $id) > 3;
                    Left(¶; not IsEmpty($ExceptionsList)) & $id)];
                ""
                )]
        Else
        Delete Portal Row [No dialog]
        Set Variable [$deleteCount; Value: Let($deleteCount = (Get(LastError) = 0) +
            $deleteCount; "")]
        End If
    End If
End Loop
Set Field [SyncConnector::gPivot; $SyncLogID]
Set Field [DeployedSyncLog::Result; $modCount & " records modified, " & $createCount &
" records created and " & $deleteCount & " records deleted"]
Commit Records/Requests [Skip data entry validation; No dialog]
```

In this sample script, rather than use multiple separate Set Variable steps, the variables used in this portion of the script are set with a Let statement inside the Exit Loop If step. For each loop, the first value of $IDs is removed. When there are no remaining values in the list (i.e. the individual value, $id, is empty), the loop will exit. Of course, if you are using table-specific $ID variables, be sure to change $IDs to the correct variable name throughout the above steps.

> **Note:** Setting the fields may be accomplished using “indirection,” using the Set Field By Name[ ] script step, in which case you will need to calculate both the field name to be set and the value. This will create a more compact script, as the script need not include separate Set Field steps for each table.

The Set Field step (after the If step that compares the deployed record’s modification timestamp to the hosted modification timestamp) attempts to set a value into the Company field. This will open (and lock) the record if it is not locked, or return error 301 (“Record is in use by another user”) if the record is locked. You may hard-code the number of attempts to modify the record, or set the number in a field in a utility or system table. If you set the number in such a field, store the value in a variable when the script runs, so that it is pulled only once for the entire sync process.

> **Note:** As long as the script is comparing each deployed record’s modification timestamp to the hosted record’s modification timestamp, you have the option to modify the records in the other direction. For instance, if a deployed record’s modification timestamp is earlier than a hosted record’s modification timestamp, you may assume that the hosted record was modified after the deployed record. If your business rules state that the hosted record “wins” in such cases, you may add steps to the sync script to set the deployed record’s fields to the hosted record’s values. The set field actions may be hard coded to the specific table, or may be calculated using Set Field By Name[ ], as noted above.

Finally, the script tests to see if the record should be deleted, goes to the appropriate portal row and deletes the portal row. Be sure to add portals to the layout, with each portal corresponding the tables you would like to delete records from. Each portal will require an object name, which you can specify in the Position tab of the Inspector in layout mode.

To move records to an intermediary table, remove the If statement testing for error 301, and the Set Variable, Else and End If steps connected with it, and replace the italicized steps with the following steps:

```
        Set Field [HostedIntermediateCompanies::CompanyID;
            DeployedCompanies::CompanyID]
        Set Field [HostedIntermediateCompanies::Company;
            DeployedCompanies::Company]
        Set Field [HostedIntermediateCompanies::Location;
            DeployedCompanies::Location]
```

Again, the Set Field By Name[ ] step can be used to create records in intermediary tables so you can avoid the necessity to separately specify each target field.

Identifying Records to be Deleted

Because the above portion of the script examines only records that have been modified since the last sync, it does not take into account records that may have been deleted in the hosted file but not modified in the deployed file. If your sync process is separated into a data exchange step and a processing step, you may configure the "exchange" portion to pass a list of records to be deleted. The list may be calculated using the Set Variable[ ] step below, and the subsequent steps may be integrated into the "processing" portion.

If the exchange and processing steps are part of the same script, the following steps identify and delete those records that have not been modified since the last sync, but have been deleted from the corresponding system. This sample uses the same model as in the script above, running from the context of the connector file's system table and using named portal objects on the script context's layout in the connector file to delete records:

```
Set Field [SyncConnector::gPivot; ""]
Set Field [DeployedSyncLog::Action; "Identify records to be deleted"]
Set Variable [$SyncLogID; Value: SyncConnector::gPivot]
Commit Records/Requests [Skip data entry validation; No dialog]
#Collect the IDs of deployed records created before the last sync timestamp that have no
corresponding hosted record.
Set Variable [$IDs; Value: Value: ExecuteSQL("Select CompanyID from DeployedCompanies
    where zGenStamp < ? and not exists (select * from HostedCompanies where
    DeployedCompanies.CompanyID=HostedCompanies.CompanyID)"; ""; ¶;
    $SyncTimeStamp)]
Loop
    Exit Loop If [Let([
        $id = GetValue($IDs; 1);
        $IDs = RightValues($IDs; ValueCount($IDs) - 1)];
        IsEmpty($id)
        )]
    Set Field [SyncConnector::gPivot; $id]
    #Be sure named portals are on the current layout
    Go to Object [Object Name: DeployedCompaniesPortal]
    Go to Portal Row [Select; First]
    Set Field [DeployedCompanies::Company; DeployedCompanies::Company]
    If [Get(LastError) = 301]
        Set Variable [$variables; Value: Let([
            $IDs = $IDs & If(PatternCount($LockedIDs; $id) < 3; Left(¶; not IsEmpty($IDs)) &
                $id);
            $LockedIDs = $LockedIDs & Left(¶; not IsEmpty($LockedIDs)) & $id;
            $ExceptionsList = $ExceptionsList & If(PatternCount($LockedIDs; $id) > 3; Left(¶;
                not IsEmpty($ExceptionsList)) & $id)];
            ""
            )]
    Else
        Delete Portal Row [No dialog]
        Set Variable [$deleteCount; Value: Let($deleteCount = (Get(LastError) = 0) +
            $deleteCount; "")]
    End If
End Loop
Commit Records/Requests [Skip data entry validation; No dialog]
Set Field [SyncConnector::gPivot; $SyncLogID]
Set Field [DeployedSyncLog::Result; $deleteCount & " records deleted"]
Commit Records/Requests [Skip data entry validation; No dialog]
```

Reducing the Record Lock Interval

The above steps accomplish the task of modifying a matching record by "touching" each field. This may increase the amount of time the record is locked beyond what is required if there are few modifications to be made. To reduce this time and to avoid touching fields that do not need to be modified, collect a list of fields that should be changed, along with their values. With this list, you can then set the fields using the Set Field By Name[ ] step to set only the fields specified in the list.

Processing Deployed Records

After hosted records have been modified (or modified deployed records have been pushed into intermediary tables in the hosted file), the script can then identify hosted records and modify or delete matching deployed records or create records in intermediary tables. The procedure is the same as for processing hosted records in place—either perform a Find and loop through records, or gather a list of primary keys and work with the values one at a time.

Note: If you choose to have your script perform a Find and loop through records, the same "pivot" relationships may be utilized to add or update records. At least one record is required in the ConnectorSystem table in order to create or modify records in one file from the context of a layout based on a TO that points to a table in the other file. To modify an existing record, the relevant key value must first be written to the global key field ("gPivot") in the ConnectorSystem table to establish the relationship path between the current context and the target TO.

Scheduled Scripts on the Host

If you are processing records after exchange, a modified version of the script steps above can be used to work with the intermediary tables. A script can be scheduled in FileMaker Server or a stand-alone robot system to process records in intermediary tables at intervals appropriate to your scenario. For instance, if users are syncing throughout the day, you may want the script to run every few minutes. If users will generally sync in the mornings or evenings, the script may be scheduled to run during those periods.

The script should follow a similar format to the sync script in the connector file—identify appropriate records, loop through comparing them to their counterparts in live tables, and make updates as necessary. While you may design this script to simply update live records, it is advisable to compare the modification timestamp from the record in the intermediary table to the modification timestamp of the hosted record, in case the hosted record was modified after the deployed record was moved into the intermediary table.

Processing in the Deployed File

Again, if you are processing records after exchange, a modified version of the script steps above can be used to work with the deployed tables. After records are exchanged and the connector and hosted files are closed, the script may continue on or call a separate script to process records from intermediary tables. In this case, the gModOverride mechanism should be used to ensure that the zModStamp field in each record is set to the modification timestamp stored with the record data in the intermediary table (remember that no fields in intermediary tables should be set to auto-enter, but should instead be populated with data from the source record by the sync script).

Deleting Intermediary Records

It is possible that the intermediary records may be retained to serve as a history of changes by different users, in which case your processing script should update a status field to identify them as processed, or transfer them to an archive. In all other cases, intermediary records should be deleted as soon as they have been processed.

Disconnection and Housekeeping

After records have been exchanged, and processed if that is part of the sync, it is time to create log records documenting the conclusion of either the exchange or the sync:

```
Set Variable [$EndSyncTimeStamp; Value: Get(CurrentTimeStamp)]
Set Field [SyncConnector::gPivot; $HostSyncLogID]
Set Field [HostedSyncLog::Result; “Sync by “ & DeployedSystem::UserID & “ complete at ”
& $ EndSyncTimeStamp]
Commit Records/Requests [Skip data entry validation; No dialog]
Set Field [SyncConnector::gPivot; “”]
Set Field [DeployedSyncLog::Action; “Sync complete at ” & $ EndSyncTimeStamp]
Commit Records/Requests [Skip data entry validation; No dialog]
Exit Script [Result: $CurrentSyncTimeStamp]
```

Finally, as noted in the “Exchanging Records” section, once the sync script in the connector file is finished, the *deployed* file should close the connector file and the hosted file, and then inform the user that the sync is complete.

Note: In the final notification to the user, you may choose to include information about the numbers of records that were deleted, modified, created and/or not updated. If so, be sure to collect the information during the appropriate portions of the script in the connector file, for instance in script variables that are passed back to the calling script as a script result, in global fields or in log records.

Conclusion

The options described in this paper are intended to be guides, possibly forming the basis for your own sync system and possibly giving you ideas for your own approach. The over-arching concept of sync is simple—make changes to one or more sets of records in order to align them with other sets of records. The actual process may be detailed and complex, depending on business rules and whether comparisons and modifications take place between live records or intermediary records. However, complexity can be addressed with planning and judicious use of FileMaker’s functions and features, producing a robust and reliable sync system.

Resources

FileMaker Technology Network: http://www.filemaker.com/technet/
FileMaker Training Series: http://www.filemaker.com/support/training/fts.html
FileMaker Forum: http://forums.filemaker.com/
FileMaker Consultants: http://developer.filemaker.com/search/
Commercial Solution GoZync: http://www.gozync.com
Commercial Solution MirrorSync: http://www.360works.com/mirrorsync

About the Author

Katherine Russell is senior consultant and team leader with NightWing Enterprises in Melbourne, Australia. Learn more about Katherine at:

https://fmdev.filemaker.com/people/katherinerussell