Audit with MBS FileMaker Plugin

How FileMaker Pro can log changes

Since 2012, the MBS plugin has an audit functions. With version 6.5 they got much faster and therefore you may want to take a closer look on our audit functions. In FileMaker Pro, you can use them to easily log all changes to the database. Later you can inspect who changed records and use the log to undo actions or display old values.

Audit with MBS

We start with a test database. For the example, we take the starter solution called event management. The audit function need an AuditLog table and a matching layout. Both of these should be available somewhere and can be in a different database. The MBS plugin looks into the AuditLog layout to determine which fields should be written. The records are then written internally using SQL commands. Therefore, there must be only a relationship to an AuditLog table, even if it is located in a different database. This is quite interesting for server-based solutions, if the log is split into several AuditLog tables and the relationships define which table logs into which AuditLog table.

We add fields to the AuditLog table. The following fields are required: FieldName, FieldHash, TableName and RecordID. The name for the field and the table define the field exactly. We know which record was changed via the RecordID and FieldHash is then the hash of the field value. The field value can be very long and can, but does not have to be written down.

The following fields can be defined:
FieldValue, FieldOldValue, FieldType, UserName, IP, CurrentTimestamp, TimeStamp, CurrentTime, CurrentDate, Action, CurrentHostTimeStamp, PrivilegeSetName, AccountName, LayoutNumber, ApplicationVersion, FileName, HostApplicationVersion, HostName, HostIPAddress, LayoutName, PageNumber, LayoutTableName, TableID, FieldID, ScriptName and WindowName.

In FieldValue, the plugin saves the new value. In the FieldOldValue field, it saves the old value of the field. This is, of course, only available if the plugin finds the older entry. FieldType saves the type of the field, for example text. UserName saves the user name and AccountName the current account name. The CurrentTimestamp, CurrentHostTimeStamp, TimeStamp, CurrentTime, and CurrentDate fields store all the current time and/or current date. Hostname and HostIPAddress define on which computer the change was made. LayoutName and LayoutNumber which layout was used, WindowName which window. TableID and FieldID store the IDs for the table and the field. Later you can then find values by IDs even faster.

Create an Audit table

So let's put a table called AuditLog and the fields FieldName, FieldHash, TableName, RecordID, FieldValue, FieldOldValue, FieldType, UserName, IP, CurrentTimestamp, Action, FileName and LayoutName. We also add an additional field to EventID. Because we can always save the ID of the event and then find the changes for an event. FileMaker automatically creates a layout for the new table. This layout is required by the current plugin to find the fields in the table. We can later see in the layout what was logged. Of course, regular users may not need to see the audit table or the layout.

You can create additional fields in the AuditLog table. For example, a LogTime field with the data type time stamp for the current time of logging. This field can automatically be filled by FileMaker. You can also pass these additional fields later using an audit call, for example, to log a variable.

Audit fields

In each table, we create an AuditTimeStamp field. This field is automatically set by FileMaker to the current time stamp when the record changes. If you already have a field with a different name, you can also use it and do not need a field with a redundant time stamp.

We also create a second field called AuditState. This will be a calculated value, which needs to be recalculated each time. So please uncheck the "Do not replace existing value of field" checkbox. The calculation is the call to Audit.Changed. This function has several parameters. First, we pass our time stamp, behind the name of the table itself, in this case events. Then we can specify fields that should be ignored. We can also use field name and | as a separator to fill a field with a given value. So, we set the value of the EventID field to the value of the EVENT ID

MATCH FIELD field. Finally, we pass the label from the field "Task Label Plural" as this field should not be logged. This field is calculated easily from the other fields. Thus, the audit call looks like this:

MBS ("Audit.Changed"; AuditTimeStamp; "Events"; "EventID|" & EVENT ID MATCH FIELD; "Task Label Plural");

For the other tables, the calls would be:

MBS ("Audit.Changed", AuditTimeStamp, "Contributors", "EventID|", & EVENT ID MATCH FIELD)
MBS ("Audit.Changed"; AuditTimeStamp; "Tasks"; "EventID|" & EVENT ID MATCH FIELD)
MBS ("Audit.Changed", AuditTimeStamp, "Agenda", "EventID|", & EVENT ID MATCH FIELD)
MBS ("Audit.Changed", AuditTimeStamp, "Guests", "EventID|", & EVENT ID MATCH FIELD)

So the plugin logs all the changes for the events. You can open the windows side by side, one with the events layout and one with the layout for the AuditLog table. For changes to the portals you have to click on a free space in the event to write the changes to the record. This writes the changes for all portals together. In the AuditLog, we see many new entries for the new records.

Positive or negative?

There are two Audit.Changed commands in the plugin. Audit.Changed takes the field names that are not to be observed. On the other hand, Audit.Changed2 is different and takes a positive list of fields. Basically, Audit.Changed2 is faster because it does not have to query which fields are available. Form fields or unsaved calculations are not monitored by default, since these are easy to recalculate.

With the Audit.SetIgnoreCalculations function, you can completely disable the monitoring of all calculation fields. With Audit.SetIgnoreSummaryFields, you can ignore the summary fields. Or with Audit.SetIgnoreUnderscoreFieldNames all fields with underline at the beginning of the word. You can also use Audit.SetIgnoredFieldNames to globally specify a list of field names for fields that you want to ignore. by default the plugin ignores the AuditState and AuditTimeStamp fields.

Display the audit

We can display the audit data for the record. Perhaps not for every user, but the display is always useful as a change log. To do this, create a relationship between Events::EVENT ID MATCH FIELD and AuditLog::EventID in the relationship diagram.

Now create space in the event layout and create a portal for the AuditLog table. If it is not found in the reference records pop-up menu, you have created the relationship above incorrectly. You could sort the fields down by CurrentTimeStamp. It is enough to take the fields FieldName, FieldValue and perhaps UserName.

Audit on deletion

If you want to log the deletion, you have two options. Either you let the user delete only by script and you always notify the plugin before deleting, or you use the security settings in FileMaker for a delete trigger.

The call from the plugin would then be via the Audit.Delete function:

MBS ("Audit.Delete"; MyTable::AuditTimeStamp; "MyTable")

The time stamp is passed, but not really used. Instead of using MyTable, you specify the current table from the current record. The plugin logs the record as with the other calls, so all the changes to the fields if needed and then an entry for deletion. You can of course do this in a script shortly before the delete command (with variable set).

Test users

First we need a test user to delete. In the Security dialog, we create a new user. We create a new set of permissions for this user. Best way is to duplicate and customize the permissions from the data input. In the settings for the calculation, there is a pop-up menu for records in the upper left corner. There, select "Custom privileges ...". A suitable dialog appears, where you can set the permissions for each table. For each table, select "Limited ..." instead of Yes. A new dialog box appears for the calculation of the limitation. Here, please insert the plugin call, for example for the events table:

MBS ("Audit.Delete", AuditTimeStamp, "Events")

For testing, we are building a script called login. This script gets only one command: Re-Login [With dialog: On]. If everything works out, you can use the new script to log in again with the new user. There you create a new data record. Make sure you commit some changes. You will see the recording in a second window with the AuditLog table. If you delete the record now, the calculation will be executed and the record should be marked as deleted. Please note that FileMaker first performs the calculation and then displays a dialog. If the user presses Cancel there, the plugin has already set an entry and the record still remains undeleted.

Experience

Many plugin users use audit. Because FileMaker often stores changes, each database should have an audit. Easily a record is changed or deleted and no one knows later who has made what and why. In addition, you can offer an undo feature, if you can find what was previously in a field.

The table with AuditLog is growing quickly, because a record is created for each change. However, you can easily rotate the table, for example change it monthly. The AuditLog table and layout must be available only in an open file. Therefore, it is recommended to put the AuditLog table in another database file. You can even have different audit databases for different files and separate the log data. Thus, a table can log into an audit file and another table logs to another file. For this, however, the relationships must be true, so that under the name AuditLog is a reference in the relationships that point to the corresponding table. Those audit log tables have of course different names.

The whole auditing works at best, when the users don't know about it. If an employee then makes a mistake, a supervisor can undo the change. Or if the data does not agree, who has done what and when.

In version 6.5, we have significantly improved SQL statements for audit, so the new plugin is much faster in the network. For each change, we need to look for old entries in the AuditLog and see if the values have changed.
Give it a try!