

# FileMaker Pro Latest and Greatest

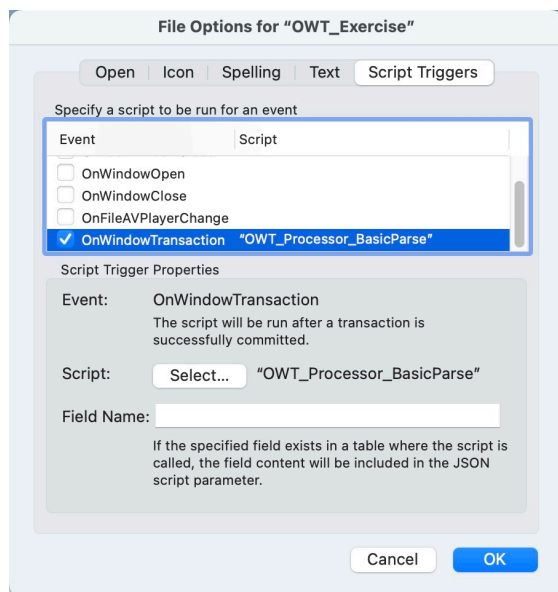
## Engage 2024 - Bob Bowers and Todd Geist

### OnWindowTransaction - Activity

Using the **OWT\_Exercise.fmp12** file, your objective is to set up the **OnWindowTransaction** trigger to log any deletes that happen within the file. The file already has many of the pieces you need, so you'll just need to modify them and get everything hooked up and tested.

#### Part A - Enable and set up the Payload calc

1. Open the File Options dialog and enable the **OnWindowTransaction** script trigger. Point it to the **OWT\_Processor\_BasicParse** script. (Leave the Field Name blank for now.)



2. Modify some data and confirm that the trigger is working. You should see a dialog with some information about the transaction you're making.
3. Go back into the File Options dialog and type in **z\_wt\_Contents** as the Field Name for the trigger.
4. Make changes to some data again and note the difference in what the dialog shows.
5. Open field definition for **Event::z\_wt\_Contents**. Modify it so that the Description and MaxAttendees fields are also part of the payload, as follows:

```
JSONSetElement ( "" ;  
[ "Description" ; Description ; JSONString ] ;  
[ "MaxAttendees" ; MaxAttendees ; JSONString ] ;  
[ "EventDate" ; EventDate ; JSONString ] ;  
[ "EventName" ; EventName ; JSONString ] ;  
[ "Cost" ; Cost ; JSONString ] ;
```

```

["ID" ; ID ; JSONString ] ;

["_User" ; Get (AccountName) ; JSONString ] ;
["_ModCount" ; Get (RecordModificationCount) ; JSONString ] ;
["_ModFields" ; Get (ModifiedFields) ; JSONString ]
)

```

6. Modify some Event data, and notice again the change in the payload shown in the dialog. Notice also that if you modify an Expense, there is no payload information.
7. Add a payload field to the **Expense** table. Be sure to make it an unstored calculation, and to uncheck the *Do not evaluate if all referenced fields* are empty option. You can decide what to include; it will probably look something like this:

```

JSONSetElement ( "" ;
  ["ExpenseName" ; ExpenseName ; JSONString ] ;
  ["Category" ; Category ; JSONString ] ;
  ["ExpenseDate" ; ExpenseDate ; JSONString ] ;
  ["Amount" ; ExpenseAmount ; JSONNumber ] ;
  ["ID" ; ID ; JSONString ] ;

  ["_User" ; Get (AccountName) ; JSONString ] ;
  ["_ModCount" ; Get (RecordModificationCount) ; JSONString ] ;
  ["_ModFields" ; Get (ModifiedFields) ; JSONString ]
)

```

8. Test that the calculation results now show up in the payload when you add/modify/delete an Expense.

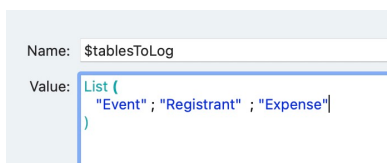
## Part B - Modify the Processor

9. Open the **Script Workspace**, and familiarize yourself with the **OWT\_Processor\_BasicParse** script. This script has been set up to loop over the JSON of the incoming transaction payload and parse it into individual transactions so it can be easily filtered and logged.

(Line 20 contains the **Show Custom Dialog** that you see anytime you commit a change. You can disable this anytime you want, though it can be helpful to leave it on for troubleshooting.)

10. Enable the **Show Custom Dialog** on line 65 of the script. Save the script, then once again modify some data. Modify an Event and several Attendees so you can see the impact of having multiple transactions in the payload.
11. The script contains table filtering logic, so that it only processes transactions from certain tables.
  - Line 13 sets a variable called **\$tablesToLog** with a list of BaseTable names.
  - Line 39 checks to see if the table referenced in the JSON payload is on that list.

Modify line 13 so that the Expense table is included in the list of tables to log.



12. Add a step to the script after line 63 that sends the **\$logData** variable as a parameter to the **Logger** script in the **DeleteLog** table.

```
59      # =====
60      # Here's where you've finally isolated a single transaction, and it's ready to process
61      # =====
62
63      Set Variable [ $logData ; Value: JSONSetElement ( "" ; [ "file" ; $myFile ; JSONString ] ; [ "table" ; $myTabl... ]
64      Perform Script [ Specified: From list ; "Logger" from file: "DeleteLog" ; Parameter: $logData ]
65
66      Show Custom Dialog [ JSONFormatElements ( $logData ) ]
67
68
69      # =====
```

13. Make some record changes, and then look at the **DeleteLog** file to confirm that it's recording each transaction. (The **DeleteLog** file will probably be open as a hidden window, so you'll need to use the **Window > Show Window** menu to unhide it.)
14. At this point, you may want to disable the two **Show Custom Dialog** steps in the processor script.
15. To make it so only Delete transactions are logged, add If / End if steps around the call to the secondary processor. Those steps should only be performed if the \$action is "Deleted".

```
59      # =====
60      # Here's where you've finally isolated a single transaction, and it's ready to process
61      # =====
62
63      If [ $action = "Deleted" ] {fx
64          Set Variable [ $logData ; Value: JSONSetElement ( "" ; [ "file" ; $myFile ; JSONString ] ; [ "table" ; $my... ]
65          Perform Script [ Specified: From list ; "Logger" from file: "DeleteLog" ; Parameter: $logData ]
66
67          // Show Custom Dialog [ JSONFormatElements ( $logData ) ]
68      End If
69
70      # =====
```

16. Spend a few minutes adding/editing/deleting data and confirm that only the deletes are being logged.

---

## Extra Credit:

The file contains two other processor scripts. Modify the **OnWindowTransaction** trigger to call those and explore the results.

- **OWT\_Processor\_FullPayload** does no parsing or filtering and just passes the whole payload along to be logged.
- **OWT\_Processor\_FullAuditLog** is similar to the basic parse, except that it sends transactions to a different secondary processor where they are broken down to the field level and old/new values are logged. (New values are checked against previous log entries to determine if a change has been made.) You can then use the magnifying glass icons on the Event Detail layout to see changes to that record / field / portal row.