FMK | FileMaker Konferenz

Überarbeitete Programmiertechniken in FileMaker

Dr. Adam G. Augustin

www.agametis.de

Wer bin ich?

- Selbständiger FileMaker Entwickler im Raum München
- Beratung und Entwicklung seit über 10 Jahren
- Entwicklung von kundenspezifischen Datenbanken sowie Betreuung und Weiterentwicklung bestehender Lösungen
- FileMaker 12 bis 18 zertifiziert
- Zahlreiche Vorträge auf der FMK und dotfmp
- Web- und App-Entwicklung
- Mehr zu meiner Philosophie auf www.agametis.de

Ziel des Vortrages

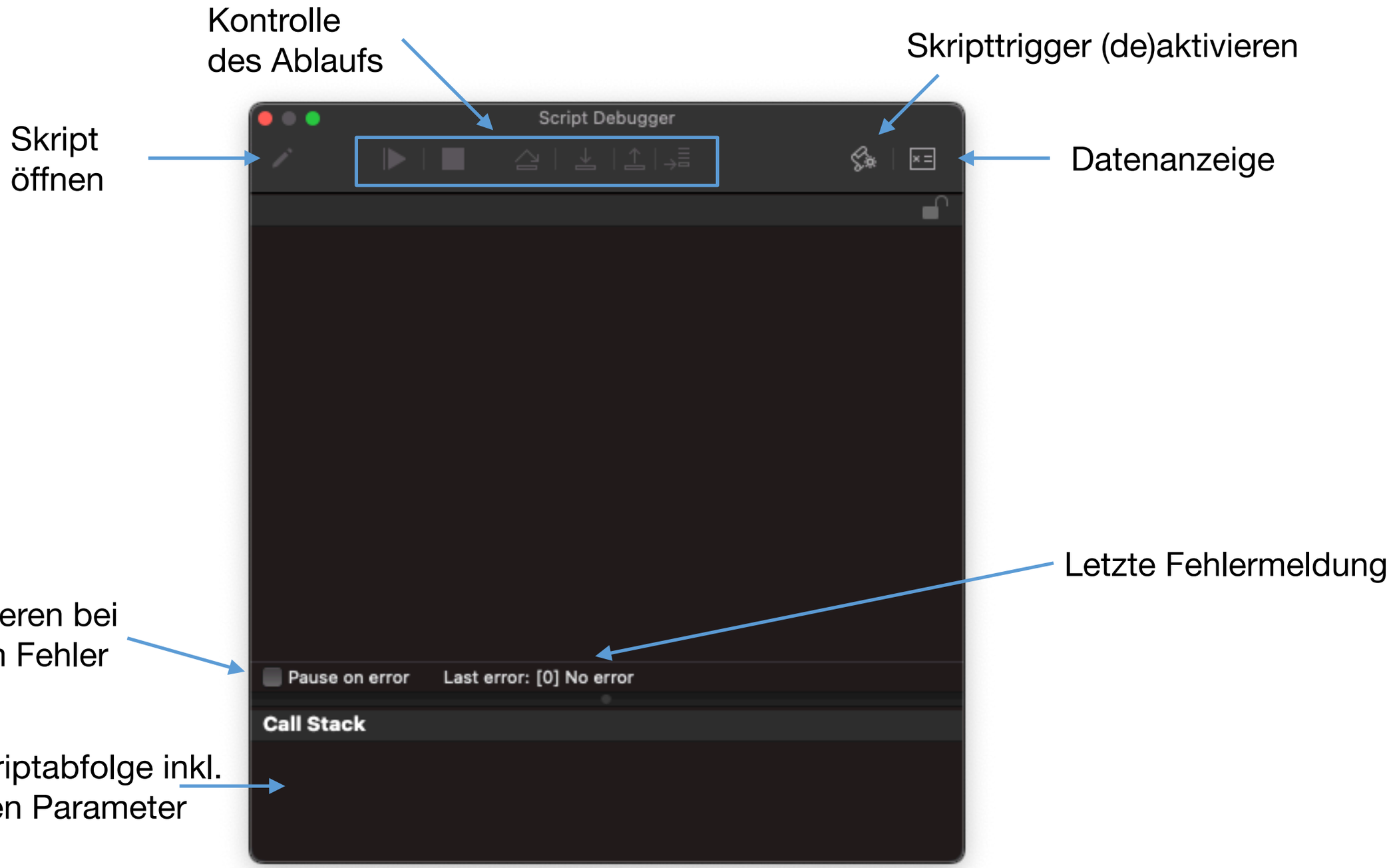
- FileMaker hat uns über die Jahre immer wieder neue Möglichkeiten gegeben, Prozesse zu programmieren und zu gestalten.
- Im Laufe meiner Arbeit habe ich mit vielen unterschiedlichen Entwicklern zusammengearbeitet und in vielen “Fremd”-Lösungen einiges Gutes und nicht so Gutes gesehen.
- Grundsätzlich ist es immer gut, seine Arbeitsweise zu überdenken und zu schauen, ob die neuen Möglichkeiten einem helfen, besser und effizienter zu entwickeln.

Der Script-Debugger

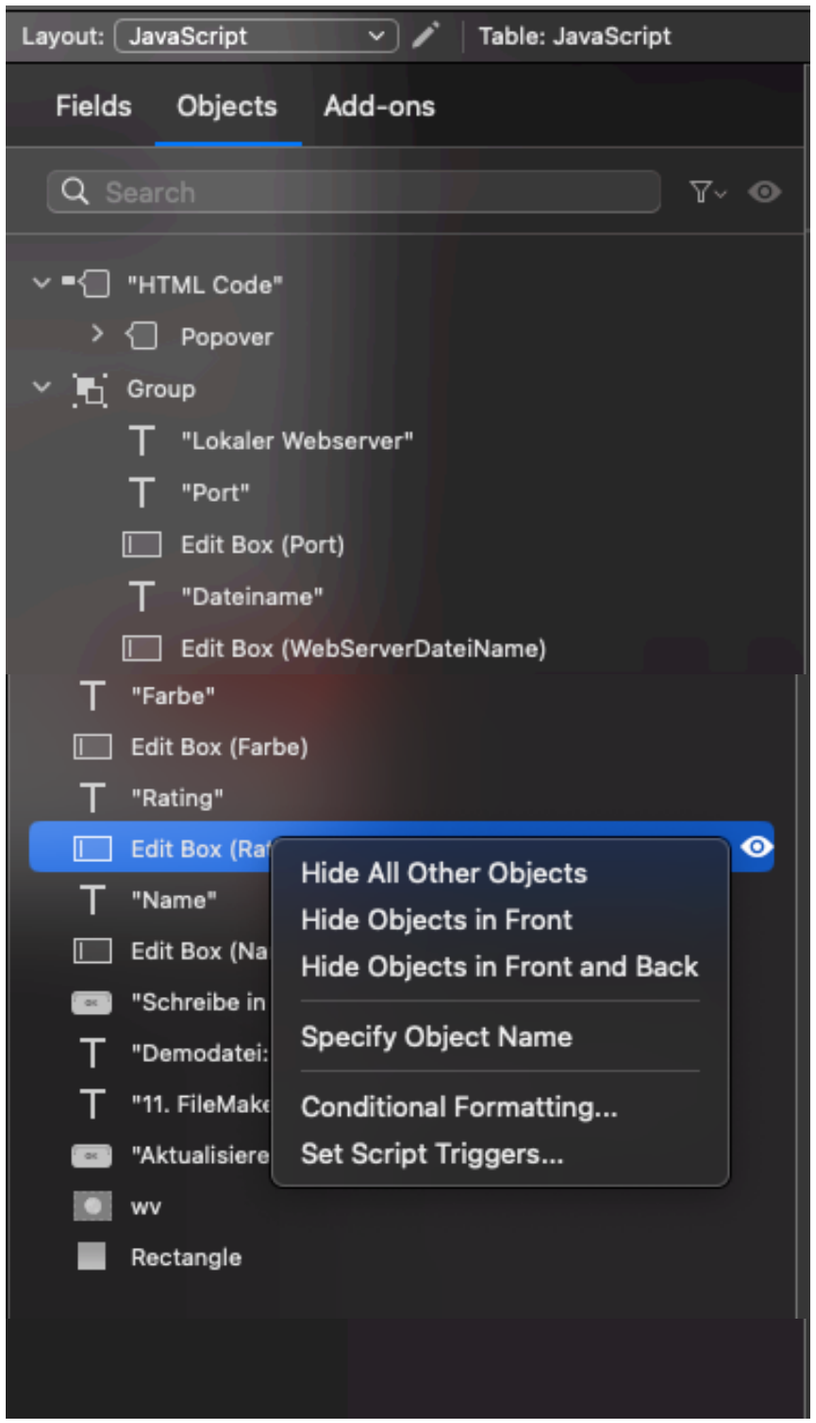
- Gibt Einsicht in den Skriptablauf
- Erlaubt die Überwachung von Variablen und dynamischen Feldinhalten
- Erlaubt die Analyse mit verschiedenen Benutzerrechten!
- Es können Haltepunkte definiert werden
- In Kombination mit der Datenanzeige (Data Viewer) bekommt man einen sehr guten Gesamtüberblick.
- Kann im Falle eines Fehlers automatisch eingeblendet werden
- Erlaubt temporär alle Skripttrigger zu deaktivieren

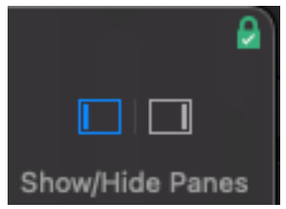
https://help.claris.com/de/pro-help/content/debugging-scripts.html

Debugger: Fenster



Objektfenster

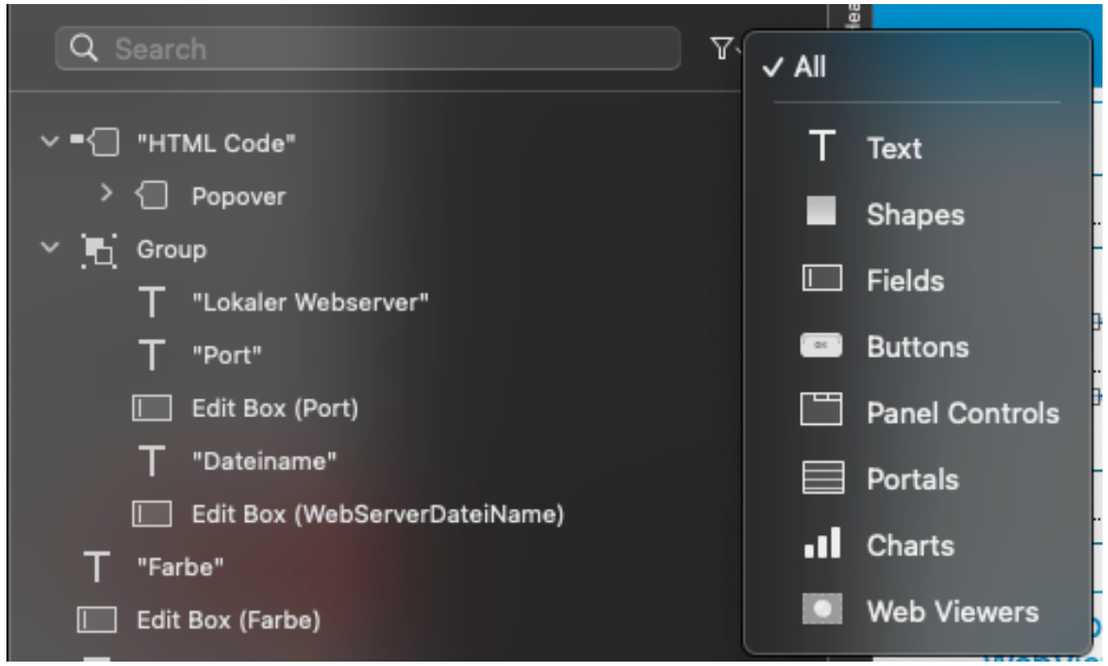




Rechts oben im Fenster befindet sich das Icon zum Ein- und Ausblenden

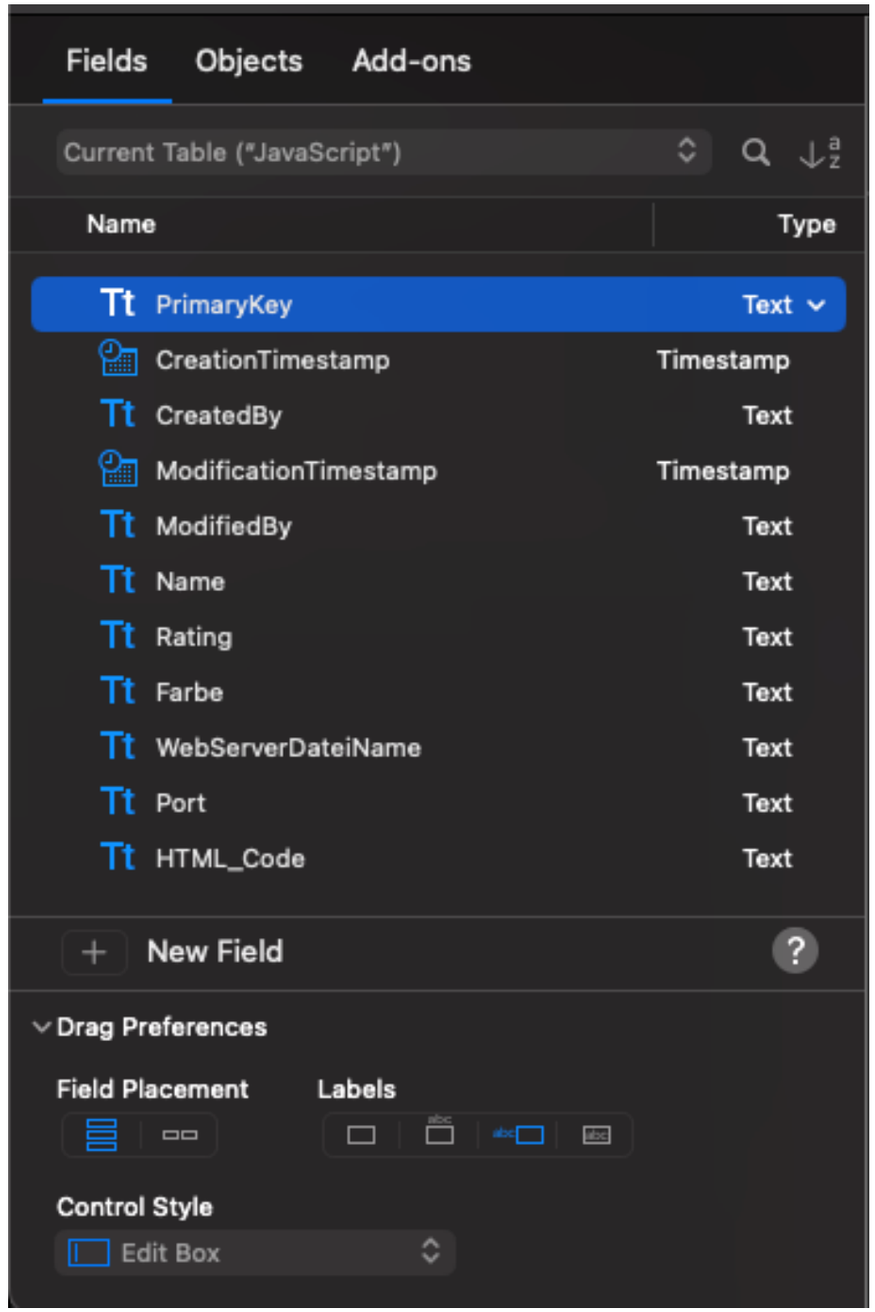
Objektfenster

- Ein- und Ausblenden von Objekten/Objektgruppen
- Definition des Objektnamens
- Definition bedingter Formatierung und Skripttrigger
- Suche und Filtern von Objekten



Objektfenster - Felder

- Felddefinition und Organisation
- Viele hilfreiche Möglichkeiten, Felder im Layout per Drag&Drop zu platzieren
- Filtern nach Feldnamen



Funktion: Liste

- Erlaubt bequemes Datensammeln (innerhalb oder über Skripte hinweg)
- In Kombination mit der Funktion “Austauschen” (substitute) bequeme Möglichkeit auf einfache Art, komplexe Strukturen aufzubauen
  - Einfaches Beispiel:
    - $Liste = Liste ($Titel ; $Vorname ; $Nachname)
    - $VollstaendigerName = Austauschen ($Liste ; “¶” ; “ “)
    - “ “ wird nur gesetzt, wenn tatsächlich die Werte vorhanden sind! (yes)
    - Keine komplizierten Überprüfungen nötig, ob $Titel, $Vorname oder $Nachname leer sind, um zu verhindern, dass unnötige Leerzeichen angehängt werden

$Titel
$Vorname
$Nachname

- Frage: Wie baut ihr eine Anschrift zusammen?

Funktion: Liste

Aufbau einer Anschrift mit Hilfe der Funktionen “Liste” und “Austauschen”

```
Let([

        xFirma = Adresse::Firma;
        xSonder = Adresse::SondervermoegenFonds;
        xStrasse = Adresse::Strasse;
        xPLZOrt = Substitute ( List ( Adresse::Plz ; Adresse::Ort ) ; “¶” ; “ “ );
        xLand = Case ( PatternCount ( Adresse::Land ; "Deutsch" ) ; “” ; Adresse::Land )

];

        List ( xFirma ; xSonder ; xStrasse ; xPLZOrt ; xLand )

) // End Let
```

Skriptschritt: Text einfügen

- Erlaubt die Verwendung von Texten ohne “Sonderzeichen” schützen zu müssen (“ \)
- Text ist direkt im Skript verwendbar.
- Text muss nicht aus einem Feld gezogen werden.

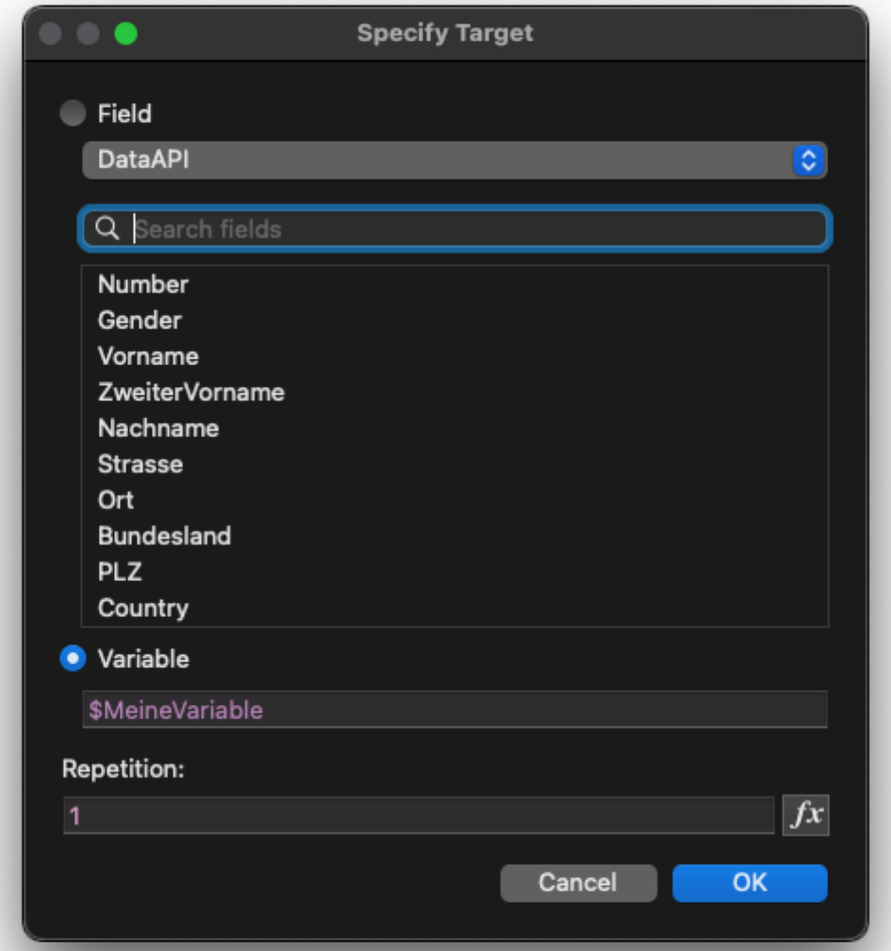
```
        <label for="farbe">Farbe:<br></label>
        <input id="farbe" type="color">
    </div>
    <button onclick="DatenSenden()">Senden</button>
</body>

<script>
    function DatenSenden() {
        var name = document.getElementById("name").value;
        var rating = document.getElementById("rating").value;
        var farbe = document.getElementById("farbe").value;
        var param = name + '\n' + rating + '\n' + farbe;
        FileMaker.PerformScriptWithOption("SchreibeWebViewerDatenInFeldern", param, 5);
    }
```

Beispiel eines Textes, welcher nicht ohne Weiteres im Formelfenster verwendet werden kann.

Skriptschritt: Text einfügen

- Seitdem es die Möglichkeit gibt, das Ergebnis in eine Variablen zu schreiben, ist der Skriptschritt viel flexibler einsetzbar bei:
  - Entwicklung für den Web Viewer (HTML, CSS und JavaScript)
  - Texte in denen “Sonderzeichen” verwendet werden



Kartenfenster (Eigenschaften)

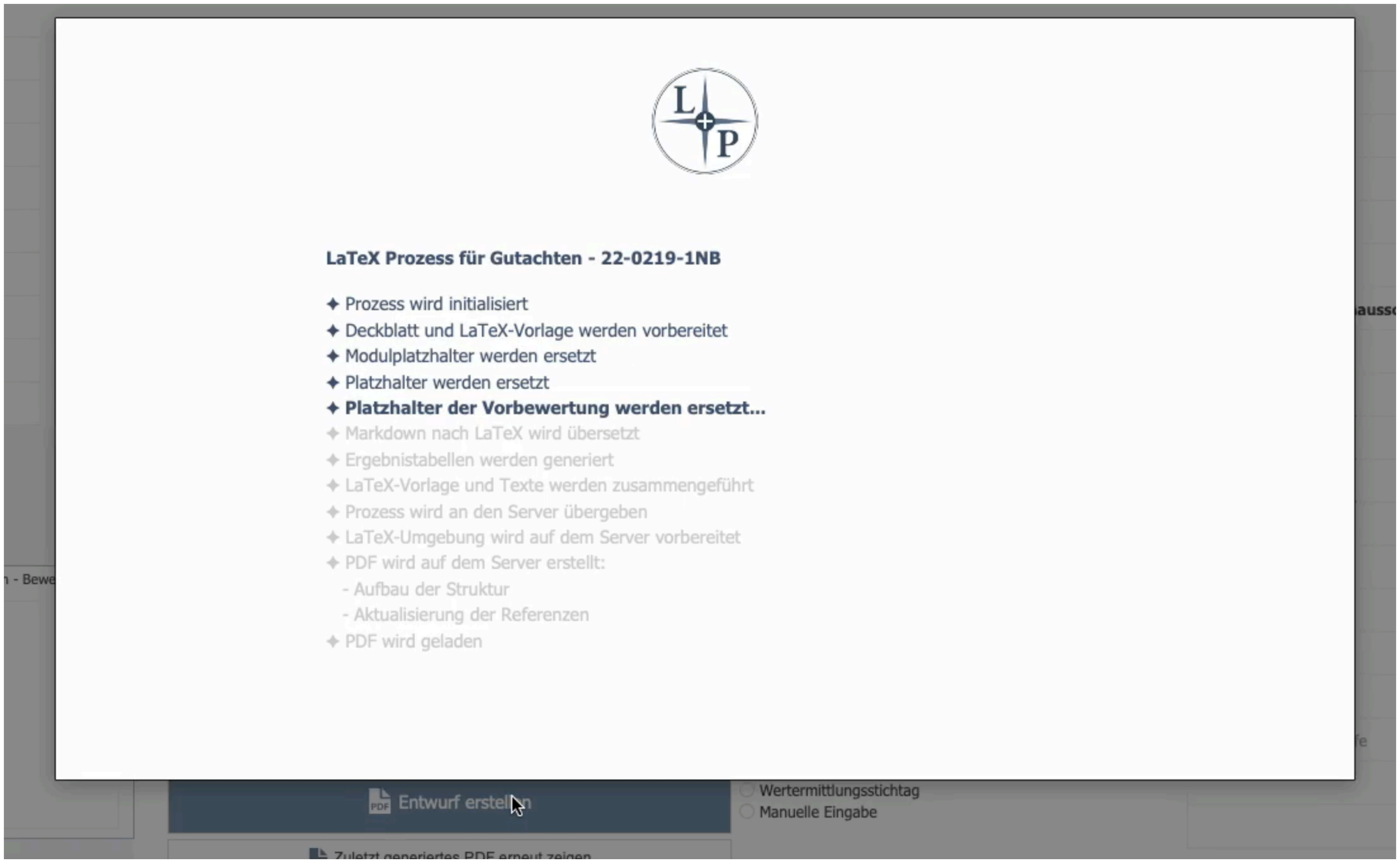
- Modales Fenster innerhalb des aktuellen Fensters
- Ohne Fensterrahmen und beliebig platzierbar
- Beim Öffnen gibt es kein “Flackern” des Hauptfensters unter Windows.
- Vorsicht: aus dem Kartenfenster heraus kann kein weiteres Kartenfenster mehr geöffnet werden (alle anderen Fenstertypen schon).

Kartenfenster (Einsatzmöglichkeiten)

- Aufbau moderner Benutzeroberflächen möglich
  - Dateneingabe mit speziellen Vorgaben
  - Modulares Suchfenster
  - Navigationslayout bei Bedarf einblendbar
  - Komplexe Aktionen mit vielen Skripten können in einem (bei Bedarf sogar sichtbaren) Fenster abgearbeitet werden, ohne eventuell in jedem Skript ein neues “unsichtbares” Fenster zu öffnen und wieder zu schliessen.
  - Bei längeren Aktion kann es z.B. als Infofenster verwendet werden

Kartenfenster

Beispiel: Kartenfenster als Infofenster mit Anzeige des Prozessfortschrittes



“Der Schleifendschungel”

```
#
If [ $Bedingung_A ]
  #
Else If [ $Bedingung_B ]
  #
  If [ $Bedingung_C ]
    #
  Else If [ $Bedingung_D1 ]
    #
    If [ $Bedingung_E ]
      #
    Else If [ $Bedingung_F ]
      #
    End If
    #
  Else If [ $Bedingung_D2 ]
    #
  End If
  #
End If
#
```

“Globale Schleife” (try-catch)

- Alle Aktionen werden innerhalb einer Loop-Anweisung ausgeführt (try).
- Bei einem Fehler wird zunächst nur die Loop verlassen.
- Nach dem Verlassen der Loop wird der Fehler verarbeitet (catch).
- Vorteil:
  - Bei intensiver Fehlerprüfung kein ständiges: wenn Fehler - dann "schreibe Log” - eventuell “schliesse Fenster” - dann “verlasse Skript”, etc.
  - Fehlerbehandlung nur an einer Stelle: am Ende des Skriptes
  - Wenn kein Fehler auftritt, wird das Skript normal beendet.
- Nachteil:
  - Man muss sich immer bewusst sein, dass sich quasi der gesamte Code innerhalb einer Loop-Anweisung befindet:
    - “Script Verlassen” ist da nicht (macht den Ablauf “kaputt”)
    - Nur “Loop Verlassen” darf benutzt werden :)

Prinzip: Globale Schleife

try

```
# ####################
# # Globale Loop
# ####################
#
# Loop - global
Loop
  #
  # Anweisung_A
  If [ $LastError ]
   Exit Loop If [ True ]
  End If
  #
  # Anweisung_B
  If [ $LastError ]
    Set Variable [ $FehlerMeldung ; Value: "Fehler bei Anweisung_B" ]
    Exit Loop If [ True ]
  End If
  #
  #
  Exit Loop If [ True ]
End Loop
# End Loop - global
#
#
#
# ####################
# # Fehlerbehandlung
# ####################
#
```

catch

```
If [ $LastError ]
  # Fehler behandeln
  #
  If [ False ]
  Else If [ $FehlerMeldung ≠ "" ]
    Show Custom Dialog [ "Fehler: " & $LastError ; $FehlerMeldung ]
  Else
    # Infodialog
    Show Custom Dialog [ "Fehler: " & $LastError ; "Es ist der Fehler " & $LastError & " aufgetreten!" ]
  End If
  #
End If
#
Exit Script [ Text Result: cf_ResultSet ( "LastError" ; $LastError ; 1 ) ]
#
#
```

```
# Loop - global
Loop
  #
  # Anweisung_A
  If [ $LastError ]
    Exit Loop If [ True ]
  End If
  #
  # Anweisung_B
  If [ $LastError ]
    Set Variable [ $FehlerMeldung ; Value: "Fehler bei Anweisung_B" ]
    Exit Loop If [ True ]
  End If
  #
  #
  Exit Loop If [ True ]
End Loop
# End Loop - global
```

Prinzip: Globale Schleife

try

```
# ####################
# # Globale Loop
# ####################
#
# Loop - global
Loop
  #
  # Anweisung_A
  If [ $LastError ]
   Exit Loop If [ True ]
  End If
  #
  # Anweisung_B
  If [ $LastError ]
    Set Variable [ $FehlerMeldung ; Value: "Fehler bei Anweisung_B" ]
    Exit Loop If [ True ]
  End If
  #
  #
  Exit Loop If [ True ]
End Loop
# End Loop - global
#
#
#
# ####################
# # Fehlerbehandlung
# ####################
#
If [ $LastError ]
  # Fehler behandeln
  #
  If [ False ]
  Else If [ $FehlerMeldung ≠ "" ]
    Show Custom Dialog [ "Fehler: " & $LastError ; $FehlerMeldung ]
  Else
    # Infodialog
    Show Custom Dialog [ "Fehler: " & $LastError ; "Es ist der Fehler " & $LastError & " aufgetreten!" ]
  End If
  #
End If
#
Exit Script [ Text Result: cf_ResultSet ( "LastError" ; $LastError ; 1 ) ]
#
#
```

catch

```
# ####################
# # Fehlerbehandlung
# ####################
#
If [ $LastError ]
  # Fehler behandeln und Log schreiben
  #
  If [ False ]
  Else If [ $FehlerMeldung ≠ "" ]
    Show Custom Dialog [ "Fehler: " & $LastError ; $FehlerMeldung ]
  Else
    # allgemeiner Fehlerdialog
    Show Custom Dialog [ "Fehler: " & $LastError ;
      “Es ist der Fehler " & $LastError & " aufgetreten!" ]
  End If
  #
End If
```

Let’s go

Demo

zum Kartenfenster mit globaler Schleife
(mit DSGVO konformen Daten!)

JSON Wer ist eigentlich Jayson?

- “Leichtes”, strukturiertes Datenaustauschformat
  - kleine Datenmenge, kein zusätzlicher Overhead wie z.B. beim XML-Format
- Seit FM16 in FileMaker integriert:
  - JSONDeleteElement
  - JSONFormatElements
  - JSONGetElement
  - JSONListKeys
  - JSONListValues
  - JSONSetElement
  - JSONGetElementType (FM19.5)

https://www.json.org/json-en.html

JSON

- Quasi-Standard für den Datenaustausch im Web
- Ermöglicht strukturiertes Datenhandling:
  - Datenaggregation und Auswertung
  - Parameterübergabe (Multiparameter)
- Bereits in vielen Vorträgen auf der FMK präsentiert

https://www.json.org/json-en.html

Skriptschritt: FileMaker Data API ausführen

- Neue Schnittstelle, um Daten im JSON-Format innerhalb einer FileMaker-Lösung abzurufen
- Eigentlich ein Server-Feature (REST API)
  - Aber mit Hilfe des Skriptschrittes “FileMaker Data API ausführen”, kann sie direkt im FM-Client verwendet werden
- Der Aufruf kann quasi “kontextlos” benutzt werden (ohne eine aktive Beziehung zum aktuellen Kontext)
  - Es wird nur ein Layout angegeben.
  - Es werden die Daten aller auf dem Layout vorhandenen Felder (inkl. der Bezugsdatensätze!) ausgegeben.

https://help.claris.com/de/pro-help/content/execute-filemaker-data-api.html

Skriptschritt: FileMaker Data API ausführen

- Die Daten werden direkt im JSON-Format zurückgegeben
- Im Gegensatz zu der Funktion “SQL ausführen” viel performanter, da auf dem Server ausgeführt
- Detaillierte Artikel von Jörg Köster im FMM - “Das Phantom der API”
  FMM 202003-202005

https://help.claris.com/de/pro-help/content/execute-filemaker-data-api.html

Let’s go

Demo

zu “FileMaker Data API ausführen”
(mit DSGVO konformen Daten!)

Tooling

- Datenbankanalyse (nach einer Zeit fühlt sich die eigene Lösung, wie eine Fremdlösung an):
  - FM Perception (für ad-hoc Schnellanalyse)
  - CrossCheck (für tiefergehende Analyse aufgrund detaillierter Error Reports)
    - FM v19.5 kompatibel
    - mit neuen coolen Features in Version 11:
      - Zusammen mit “Save a Copy as XML…” können viele zusätzliche Informationen und Abhängigkeiten angezeigt werden (z.B: wichtige Details zu Layoutbereichen)
      - Grafische Anzeige der Skriptabfolge und Positionen von Layoutobjekten innerhalb eines Layout

Tooling

- Russell Watsons Wundertüte: https://github.com/mrwatson-de
- Clip Manager (auf Apple Silicon ist noch mit FM v18 nutzbar)
- MBS (spätestens jetzt muss man einen Mac für die FM-Entwicklung nutzen :))

Was gibt es sonst?

- UUID-Funktionen (als Text oder Zahl) z.B. für Schlüsselfelder
- "Skript auf Server ausführen” (seit 19.5 auch auf dem Server einsetzbar, um parallele Verarbeitung zu erreichen)
- Ausschnitt “auf sich selbst” - damit werden Layouts im Master-Detail-Stil möglich
- Seitensteuerelement, Tastenleiste
- Datei-Skriptschritte (https://help.claris.com/de/pro-help/content/files-script-steps.html)
- Standardfelder in Tabellen (https://support.claris.com/s/article/Default-Fields?language=en_US)
- “JavaScript in Web Viewer ausführen” (Details in meinem anderen Vortrag von der FMK2022)
- Und, und, und

https://support.claris.com/s/article/FileMaker-Pro-Version-Comparison?language=en_US

Download

https://ag.amet.is/fmk2022

F & A

Vielen Dank für euer Interesse!

Vielen Dank unseren Sponsoren

FileMaker Magazin

standpunkte

gofilemaker.de
FileMaker Software & News

JK SOLUTION

kursivsoftware

karo.at
FileMaker Development