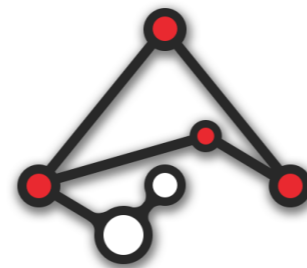


# Datenaustausch zwischen FileMaker und WebViewer

## JavaScript mit FileMaker einsetzen

Dr. Adam G. Augustin



[www.agametis.de](http://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 zertifiziert
- Zahlreiche Vorträge auf der FMK und dotfmp
- Web- und App-Entwicklung
- Mehr zu meinen Projekten mit Arbeitsbeispielen auf [www.agametis.de](http://www.agametis.de)



# Inhalt

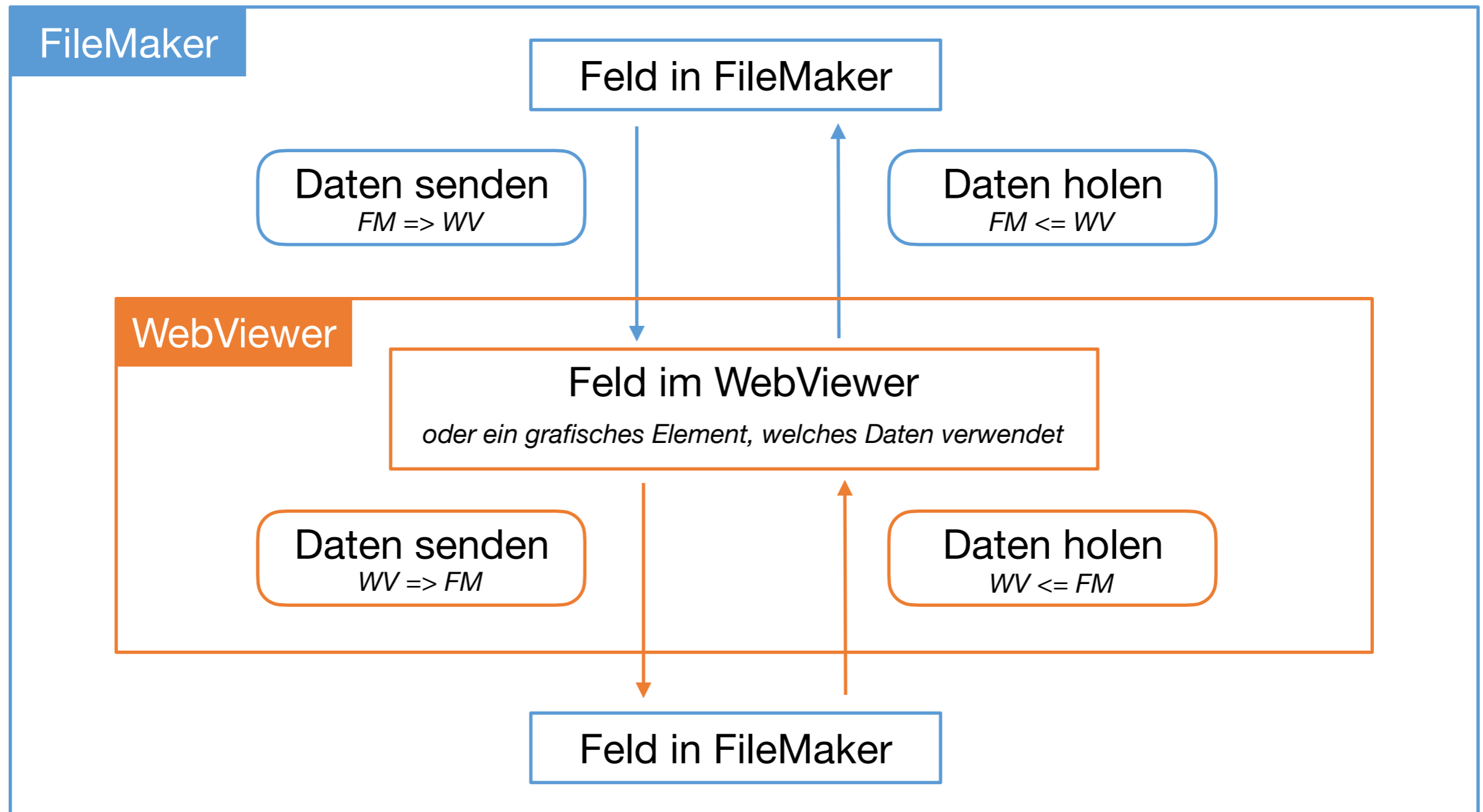
- Wie und wann können Daten zwischen FileMaker und WebViewer fließen?
- Mögliche Ansätze für die Umsetzung
- Demo am Beispiel eines Kalenders (fullcalendar)
- FAQ

The screenshot shows a calendar for October 2024. At the top, there are navigation buttons: a left arrow, 'Heute' (Today), and a right arrow. The title 'Oktober 2024' is centered, and 'Monat' (Month) is on the right. The calendar grid has columns for days of the week (Mo, Di, Mi, Do, Fr, Sa) and rows for weeks (KW40, KW41, KW42, KW43). Events are represented by green bars: 'Test auf dem iPad' (Oct 1-2), 'Fullcalendar im Test' (Oct 7-9), 'Test im WebDirect' (Oct 10-11), and 'Mehrtägiger Termin' (Oct 15-18). A light green background covers Oct 2-4, labeled 'FMK 2024'.

	Mo	Di	Mi	Do	Fr	Sa
KW40	30	1	2	3	4	
	Test auf dem iPad					
KW41	7	8	9	10	11	
	Fullcalendar im Test			Test im WebDirect		
KW42	14	15	16	17	18	
		Mehrtägiger Termin				
KW43	21	22	23	24	25	

# Wie können Daten fließen?

# Mögliche Datenflüsse und Trigger



Senden und Holen können z.B. durch Knöpfe (sowohl in FM als auch in WV) oder Interaktionen mit grafischen Elementen im WV ausgelöst werden.

# FileMaker => WebViewer

Daten werden zum WebViewer geschickt

- Mit Hilfe des Skriptschrittes **Perform JavaScript in Web Viewer**
- Eine JavaScript Funktion wird aus FileMaker heraus mit Hilfe des Skriptschrittes direkt ausgeführt.
- Beim Ausführen des Skriptschrittes werden Daten an den WebViewer geschickt.
- Tipps:
  - Alle Daten in eine JSON-Variable packen, die als **ein** Parameter im oberen Skriptschritt verwendet wird.
  - JSON-Darstellung ist einfach mit dem Skriptschritt **Execute FileMaker Data API** zu bekommen.
  - Im WebViewer werden die Daten mit Hilfe der JS-Funktion **JSON.parse()** in ein JS-Objekt umgewandelt.

# FileMaker <= WebViewer

Daten werden vom WebViewer geholt

- Mit Hilfe des Skriptschrittes **Perform JavaScript in Web Viewer**
- Eine JavaScript Funktion wird aus FileMaker heraus mit Hilfe des Skriptschrittes ausgeführt, die die Daten mit Hilfe der Funktion **FileMaker.PerformScriptWithOptions()** an FileMaker sendet.
- Mit **Get(ScriptResult)** werden die Daten in FileMaker entgegengenommen.
- Tipps:
  - Am Besten alle Daten in ein JS-Objekt packen.
  - Mit der JS-Funktion **JSON.stringify()** werden die Daten in einen String umgewandelt.
  - In FileMaker sind die Daten sofort als JSON-Objekt verfügbar.

# WebViewer => FileMaker

Daten werden vom WebViewer zu FileMaker geschickt

- Mit Hilfe der JS-Funktion `FileMaker.PerformScriptWithOptions()` werden die Daten an FileMaker übergeben.
- Mit `Get(ScriptResult)` werden die Daten in FileMaker entgegengenommen.
- Tipps analog zu `FM <= WV`



# WebViewer <= FileMaker

Daten werden aus FileMaker geladen

- Mit Hilfe der JS-Funktion `FileMaker.PerformScriptWithOptions()` werden die Daten von FileMaker geholt.
- Die aufbereiteten Daten werden mit Hilfe des Skriptschrittes `Perform JavaScript in Web Viewer` zurück an den WebViewer geschickt.
- Tipps analog zu FM => WV

# Wieso ist das Bisherige wichtig?

- Bei einfachen JS-Bibliotheken ist die Handhabung straight forward. Man kann direkt die jeweiligen Skripte/Funktionen nutzen.
- In komplexen Umgebungen, d.h. bei Verwendung von UI-Frameworks wie React, Vue, Svelte und co., ist vor allem die Variante FM  $\leq$  WV (in FM aus dem WV holen) nicht offensichtlich und mit zusätzlichem Aufwand verbunden.
- Dabei ist die Herausforderung, einen “Zugang” zu der “verschlossene Welt” des UI-Frameworks zu schaffen.
- Ein Beispiel von mir für React ist verfügbar unter:

<https://github.com/agametis/fm-react-demo>

# Datenflüsse in der React-Demo umgesetzt

The screenshot displays a FileMaker interface with a WebViewer layout. The top toolbar includes 'Records' (1 Total (Unsorted)), 'Show All', 'New Record', 'Delete Record', 'Find', 'Sort', and 'Share'. The layout is set to 'WebViewer' and includes 'View As' options and a 'Preview' button. The main content area is titled 'Bidirectional communication between FileMaker and React' and features a 'Refresh' button, an 'InputField in FM', and buttons for 'Send to WV', 'Check State', and 'HTML'. A diagram illustrates data flow: orange arrows show 'WV <= FM' (WebViewer to FileMaker) and 'WV => FM' (WebViewer to FileMaker); blue arrows show 'FM => WV' (FileMaker to WebViewer) and 'FM <= WV' (FileMaker to WebViewer). A red box highlights the 'FM <= WV' flow with the text 'Das ist die Herausforderung' (This is the challenge). At the bottom, there are 'Get from FM' and 'Send to FM' buttons.

<https://github.com/agametis/fm-react-demo>

# Wann können Daten fließen?

# Wann können Daten fließen?

- Beim Initialisieren des WebViewers (beim Start der App):
  1. Daten können direkt im HTML-Code enthalten sein
  2. Daten werden nachgeladen (meine bevorzugte Methode)
- Während der Nutzung der App im WebViewer:
  - Mit dem Skriptschritt **Perform JavaScript in Web Viewer**
  - Mit der JS-Funktion **FileMaker.PerformScriptWithOptions()**

# Wann können Daten fließen?

- Beim Initialisieren des WebViewers (beim Start der App)
  1. **Daten können direkt im HTML-Code enthalten sein**
  2. Daten werden nachgeladen (meine bevorzugte Methode)
- Während der Nutzung der App im WebViewer
  - Mit dem Skriptschritt **Perform JavaScript in Web Viewer**
  - Mit der JS-Funktion **FileMaker.PerformScriptWithOptions()**

# Daten sind direkt im Code enthalten

- Der HTML-Code enthält Platzhalter (z.B. <Platzhalter-x>, <JS-Bib>, ...) für alle möglichen Informationen wie JS-Bibliotheken, CSS und Nutzdaten.
- Platzhalter werden im FileMaker Skript durch die entsprechenden Daten ersetzt.
- HTML-Code wird anschliessend im WebViewer geladen.

# Wann können Daten fließen?

- Beim Initialisieren des WebViewers (beim Start der App)
  1. Daten können direkt im HTML-Code enthalten sein
  2. **Daten werden nachgeladen (meine bevorzugte Methode)**
- Während der Nutzung der App im WebViewer
  - Mit dem Skriptschritt **Perform JavaScript in Web Viewer**
  - Mit der JS-Funktion **FileMaker.PerformScriptWithOptions()**

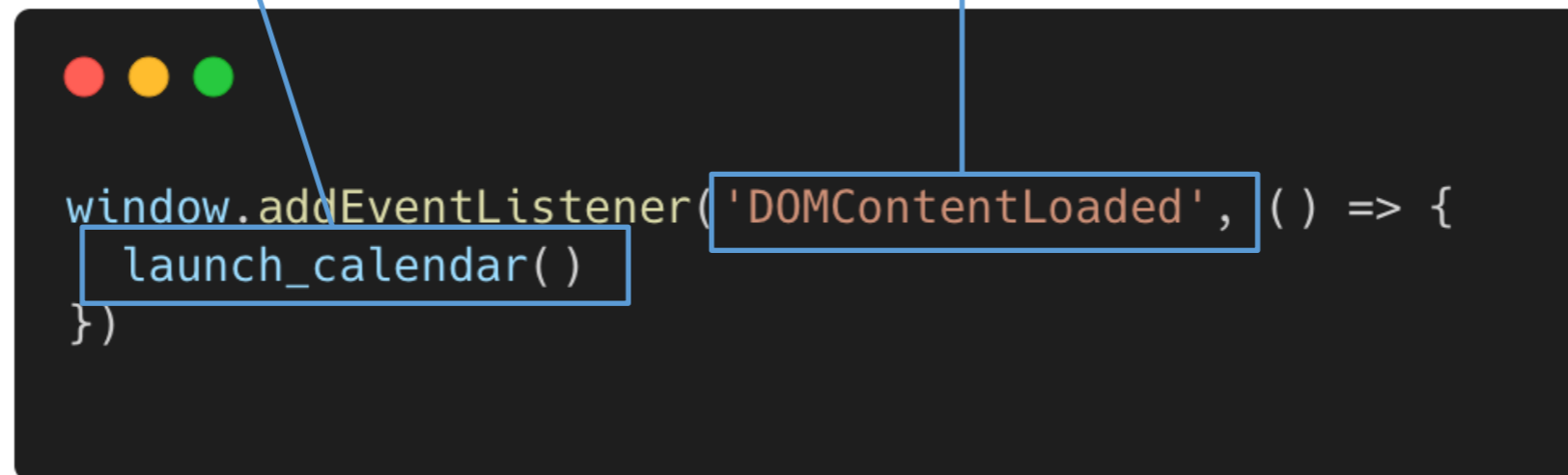


# Datenflüsse zwischen FileMaker und WebViewer am Beispiel der FullCalendar Bibliothek

Die Codebeispiele nutzen meine leicht modifizierte demo4.html aus der Demodatei von der FMK2023. Democode ist in demo4\_mod.html enthalten.

# Daten werden beim Start nachgeladen

- Sobald HTML und JS im WebViewer geladen wurden, werden sofort danach die Daten mit einer Initialisierungsfunktion nachgeladen (eigenständig ohne Benutzerinteraktion).
- Umsetzung mit Hilfe des EventListeners `DOMContentLoaded`. In meinem Beispiel wird die JS-Funktion `launch_calendar()` gestartet.



```
window.addEventListener('DOMContentLoaded', () => {  
  launch_calendar()  
})
```

The screenshot shows a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in a light-colored font. Two blue boxes highlight the event name 'DOMContentLoaded' and the function call 'launch\_calendar()'. A blue line connects the text 'DOMContentLoaded' in the second bullet point of the list above to the highlighted box in the code. Another blue line connects the text 'launch\_calendar()' in the same bullet point to the highlighted box in the code.

- Dieser Trigger wird erst dann aktiv, wenn der gesamte JS-Code geladen wurde.

[https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded\\_event](https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event)

# Daten werden nachgeladen: Holen von FM

- `launch_calendar()` wurde gestartet und “wartet” so lange, bis das Objekt FileMaker verfügbar ist.
- Danach wird das FileMaker-Skript `ext_init_calendar` aufgerufen.

```
const launch_calendar = () => {
  const interval = 10 // ms

  window.setTimeout(() => {
    let end_time = new Date()
    let time_elapsed = end_time - start_time
    console.log(time_elapsed)
    if (typeof FileMaker !== 'undefined') {
      FileMaker.PerformScriptWithOptions('ext_init_calendar', null, 5)
    } else if (time_elapsed >= 600) {
      const meldung = 'Kalender wird nur in FileMaker angezeigt!'
      document.getElementById('calendar').innerHTML = meldung
    } else {
      window.setTimeout(() => {
        launch_calendar()
      }, interval)
    }
  }, interval)
}
```

# Daten werden nachgeladen: Holen von FM

```
const launch_calendar = () => {
  const interval = 10 // ms

  window.setTimeout(() => {
    let end_time = new Date()
    let time_elapsed = end_time - start_time
    console.log(time_elapsed)
    if (typeof FileMaker !== 'undefined') {
      FileMaker.PerformScriptWithOptions('ext_init_calendar', null, 5)
    } else if (time_elapsed >= 600) {
      const meldung = 'Kalender wird nur in FileMaker angezeigt!'
      document.getElementById('calendar').innerHTML = meldung
    } else {
      window.setTimeout(() => {
        launch_calendar()
      }, interval)
    }
  }, interval)
}
```

warten

entscheiden

# Daten werden nachgeladen: Verarbeitung im WebViewer

- Das FileMaker-Skript `ext_init_calendar` ruft wiederum mit Hilfe des FM-Skriptschrittes `Perform JavaScript in Web Viewer` die JS-Funktion `init_calendar()` auf, an welche der Skriptschritt alle relevanten Daten übergibt.
- Die Parameter werden als ein JSON-Objekt an den WebViewer übergeben und mit `JSON.parse()` in ein JS-Objekt umgewandelt.
- Alle Daten sind nun vorhanden: das Kalender-Objekt wird erzeugt und in der HTML-Seite gerendert.

```
const init_calendar = (parameter) => {  
  const p = JSON.parse(parameter)  
  
  let calendarElement = document.getElementById('calendar')  
  
  calendar = new FullCalendar.Calendar(calendarElement, settings)  
  
  // Variable settings wird hier  
  // aus den Parameterdaten p zusammengesetzt  
  
  calendar.render()  
}
```

# Datenfluss während der Nutzung der App

- Damit die Kommunikation mit FileMaker-Skripten aus dem WebViewer heraus einfacher wird, gibt es eine zentrale JS-Funktion `execute_fm_script()`
- Daten werden mit `JSON.stringify()` als ein String an FileMaker übergeben.

```
const execute_fm_script = (parameter) => {  
  // Wenn option im Parameter übergeben wurde, benutze dieses, sonst nimm die 5  
  const option = parameter?.option ?? 5  
  
  const fm_script_name = parameter.scriptName  
  const fm_parameter = JSON.stringify(parameter)  
  
  if (typeof FileMaker !== 'undefined') {  
    FileMaker.PerformScriptWithOptions(  
      'ext_ausfuehren_in_filemaker',  
      fm_parameter,  
      option  
    )  
  } else {  
    alert(`FileMakerSkript ${fm_script_name} konnte nicht ausgeführt werden`)  
  }  
}
```

# Demo



# Änderungen in 2024 im Vergleich zu 2023

- Demo 2024 basiert ursprünglich auf meiner leicht modifizierten Demodatei ([demo4.html](#)) von der FMK2023.
- In der neuen Demodatei [demo4\\_mod.html](#) von FMK2024 wurden zwei Funktionen/Skripte umbenannt und eine Variable umbenannt:
  - FileMaker Skript (2023 => 2024): load\_calendar => ext\_init\_calendar
  - JavaScript Funktion: load\_calendar => init\_calendar
  - In Zeile 4 des Skriptes “ext\_init\_calendar” die Variable in “init\_calendar” umbenennen



# FAQ

Vielen Dank für euer Interesse!



[ag.amet.is/fmk2024](https://ag.amet.is/fmk2024)

# Vielen Dank unseren Sponsoren

