

JSON: Beyond the Basics

Fresh Takes, Dashboards

JSON-LD, AI Integration

Matt Navarre
Navarre Training



Who am I?



Passionate trainer, Expert developer

Certified in FM versions 7-current

Founded 3 companies

FBA Awards, conferences, podcasts,
meetups, blog, YouTube

Moved to Greece in 2022

Who am I?

Dogs & Cat

Bicycles

Musician

Conductor, singer, composer







FileMaker Champion Series

- ▶ Expert / AI Integration
 - ▶ 10 hours in class - Live online
 - ▶ 90 minutes 1 on 1
 - ▶ Build skills - not just knowledge
 - ▶ Feature-packed demo file

Agenda

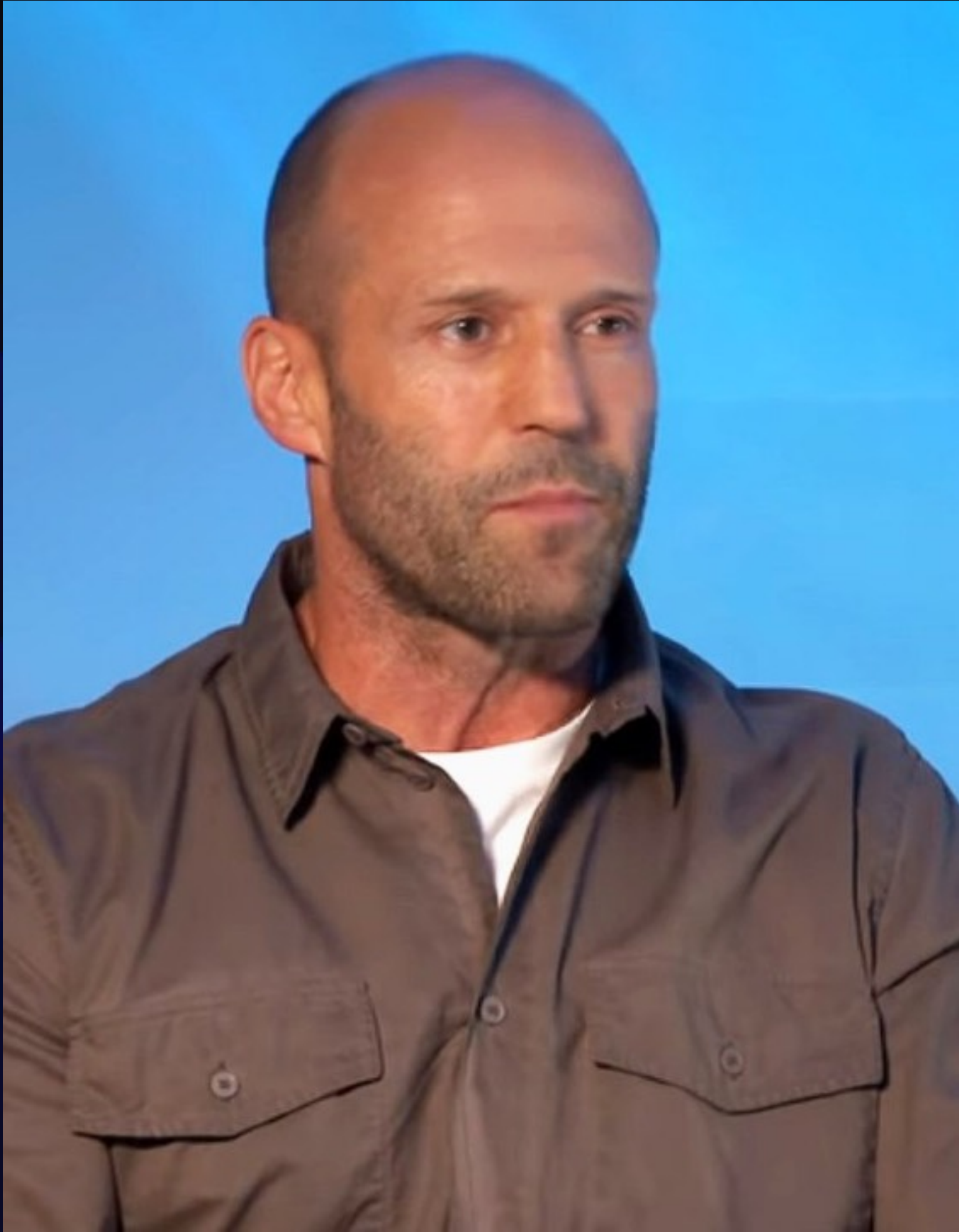
Slides ~ 15 minutes

Live demos ~ 25 minutes

Q&A ~ 15 minutes

- ▶ How do **YOU** use JSON?

JSON









Douglas Crockford
Codified & Popularized JSON

What is JSON?

JavaScript Object Notation

Structured text formatting

“Light weight data interchange format”

Human Readable

Like XML, but it doesn't suck™

What is JSON?

Hierarchical (nested structure)

VERY FLEXIBLE

FileMaker-y

Since Version 16

- ▶ May 9th, 2017
- ▶ 2.902 days ago

Why use JSON?

Simpler

- ▶ Once you are comfortable using it
- ▶ Just like Case ()
- ▶ And Let ()

Better

- ▶ Flexible
- ▶ Bulletproof

Where to use JSON?

Scripts

- ▶ Passing params in, processing params, sending JSON out

ButtonBars & Layout Calculations

Calculated Fields

Nearly everywhere you use the Calc dialog

When to use JSON?

Here

Today

Now

Always

The basics...
...a very quick review

Punctuation

FileMaker () [] ;

JSON { } , :

Functions

JSONGetElement ()

- ▶ Two arguments: Name and Value

JSONSetElement ()

- ▶ Four arguments — existing json, name, value, data type
- ▶ Ability to add or modify existing data

Functions

JSONFormatElements ()

- ▶ Makes it easier to read, but adds characters
- ▶ Rarely, it will break valid JSON (can't reproduce this today though)

JSONDelete Elements ()

- ▶ Cleans existing JSON, including dot notation

Functions

JSONListKeys ()

- ▶ Returns the names (keys) of name/value pair

JSONListValues ()

- ▶ Returns the values of a name/value pair

JSONMakeArray ()

- ▶ Newest function. Converts a List into an Array.

String, Number, Boolean

Expression:

```
JSONSetElement ( "{}" ;  
  "name1" ; "1" ; JSONstring  
)
```

Result:

```
{"name1":"1"}
```

Expression:

```
JSONSetElement ( "{}" ;  
  "name1" ; "1" ; JSONnumber  
)
```

Result:

```
{"name1":1}
```

Expression:

```
JSONSetElement ( "{}" ;  
  "name1" ; 1 ; JSONboolean  
)
```

Result:

```
{"name1":true}
```

Basics: Passing Parameters

I always use it – even for a single param

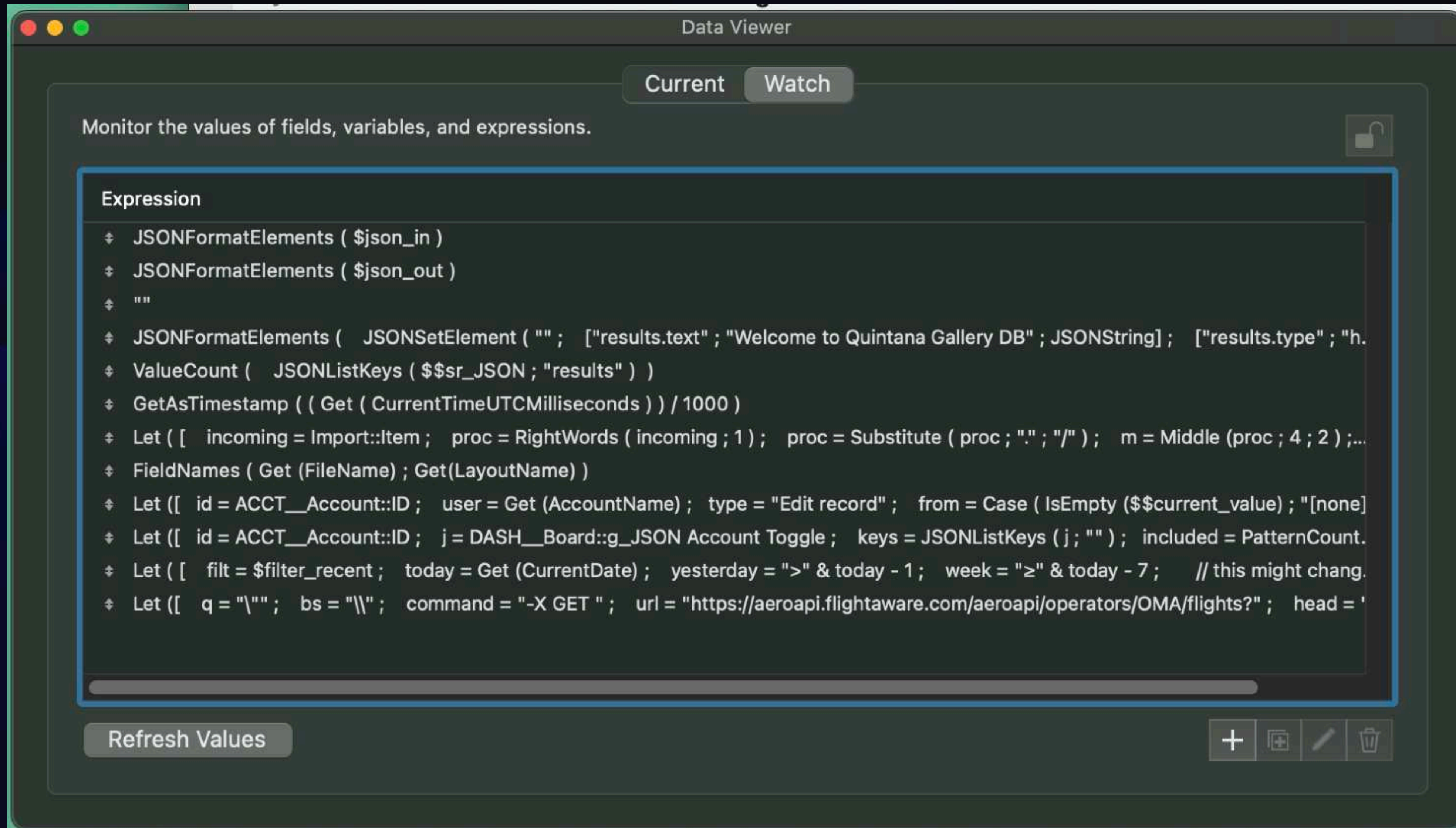
Because some day, there will likely be a 2nd param

Copy and paste code, or use shortcut

Define \$variables at top of script

Plug for MBS

If Data Viewer is your friend...



... Then ChatGPT is your secret lover

```
Optimize this JSON: Let ([
  company_array = JSONMakeArray (List (D_Company::cJSON) ; "" ;
JSONRaw)  // Must use JSONRaw here
];

JSONSetElement ("{" ;
  ["@context" ; "https://schema.org" ; JSONString] ;      // Add JSON-LD
context
  ["@type" ; "Organization" ; JSONString] ;                // Define JSON-LD type as
Organization
  ["company_list" ; company_array ; JSONArray]
)
)
```



Your original expression is solid, but here's a cleaned-up, **optimized version** of the JSON construction using proper formatting, indentation, and a slight simplification of the Let structure to make it more readable and maintainable:

filemaker

Copy

```
Let (
  company_array = JSONMakeArray ( List ( D_Company::cJSON ) ; "" ; JSONRaw ) ;

  JSONSetElement ( "{" ;
    [ "@context" ; "https://schema.org" ; JSONString ] ;
    [ "@type" ; "Organization" ; JSONString ] ;
    [ "company_list" ; company_array ; JSONArray ]
  )
)
```

Four Use Cases

Let's slow down...

1. A JSON Field in ~Every Table

2. Preferences & Session Data

3. Dashboard Data

4. JSON-LD & AI

1.

A JSON Field in Most Tables

JSON Field (stored text field)

Main Benefits

- ▶ Write any data needed without having to change schema
 - Including calculations
 - Replace multiple unstirred calc fields
- ▶ Display any data
 - Use a layout calculation or Button Bar
 - Get any element
- ▶ Not indexed

JSON Field – Uses

Store metadata about the record

- ▶ Record creation details beyond date & user
- ▶ Container data, resolution, image size

Log data

- ▶ A local list of record edits
- ▶ Record access, data quality warnings, etc.

JSON Field – Uses

Temporary fields

- ▶ Example: A special event where you need 5 fields for a month
- ▶ No need to pollute the schema

Delete unused fields – with no risk

- ▶ Append data in JSON field with `JSONSetElement ( )`
- ▶ Delete offending fields
- ▶ Celebrate

JSON Field – Notes

Enter {} in the field upon record creation

- ▶ To avoid ? Error when reading

Only use in tables where there is a high chance of use

- ▶ Or just create the field when it's needed

Not directly searchable

- ▶ Should have values set only by script

Meta- data (On Create)

A little

Expression:

L: 8 C: 70

```
JSONFormatElements (  
  JSONSetElement ( "{}" ;  
    ["created.ts" ; Get (CurrentTimestamp) ; jsonstring ] ;  
    ["created.utc" ; Get (CurrentTimeUTCMilliseconds) ; jsonstring ] ;  
    ["created.user" ; Get (AccountName) ; jsonstring ] ;  
    ["created.priv_set" ; Get (AccountPrivilegeSetName) ; jsonstring ] |  
  )  
)
```

Result:

☒ Automatically evaluate

Evaluate

```
{  
  "created" :  
  {  
    "priv_set" : "[Full Access]",  
    "ts" : "4/23/2025 11:32:18 AM",  
    "user" : "Matt Navarre",  
    "utc" : "63880997538676"  
  }  
}
```

A lot

Expression:

```
JSONFormatElements (  
  JSONSetElement ( "{}" ;  
  
    ["created.ts" ; Get (CurrentHostTimestamp) ; jsonstring ] ;  
    ["created.utc" ; Get (CurrentTimeUTCMilliseconds) ; jsonstring ] ;  
    ["created.user" ; Get (AccountName) ; jsonstring ] ;  
    ["created.priv_set" ; Get (AccountPrivilegeSetName) ; jsonstring ] ;  
  
    ["created.record_number" ; Get (RecordNumber) ; jsonnumber ] ;  
    ["created.record_id" ; Get (RecordID) ; jsonnumber ] ;  
    ["created.script_name" ; Get (ScriptName) ; jsonstring ] ;  
  
    ["modified.ts_host" ; Get (CurrentHostTimestamp) ; jsonstring ] ;  
    ["modified.ts_local" ; Get (CurrentTimestamp) ; jsonstring ] ;  
    ["modified.account" ; Get (AccountName) ; jsonstring ] ;  
    ["modified.script_name" ; Get (ScriptName) ; jsonstring ]  
  ) )
```

Result:

```
{  
  "created" :  
  {  
    "priv_set" : "[Full Access]",  
    "record_id" : 1,  
    "record_number" : 1,  
    "script_name" : "",  
    "ts" : "4/23/2025 11:39:45 AM",  
    "user" : "Matt Navarre",  
    "utc" : "63880997985433"  
  },  
  "modified" :  
  {  
    "account" : "Matt Navarre",  
    "script_name" : "",  
    "ts_host" : "4/23/2025 11:39:45 AM",  
    "ts_local" : "4/23/2025 11:39:45 AM"  
  }  
}
```

Temp
Data

/

Delete
Fields

A little

Expression:

JSONFormatElements (

JSONSetElement (C__Company::JSON ;

["Christmas Card 2017" ; C__Company::Christmas Card 2017 ; JSONString] ;

["Christmas Card 2018" ; C__Company::Christmas Card 2018 ; JSONString] ;

["Christmas Card 2019" ; C__Company::Christmas Card 2019 ; JSONString]

))

Result:

{

"Christmas Card 2017" : "",

"Christmas Card 2018" : "1",

"Christmas Card 2019" : "1"

}

A lot

Expression:

```
Let ([
  cc2017_field = "Christmas Card 2017" ;
  cc2017_value = C__Company::Christmas Card 2017 ;
  cc2017_value = Case (cc2017_value = 1 ; "Yes" ; "No" ) ;

  cc2018_field = "Christmas Card 2018" ;
  cc2018_value = C__Company::Christmas Card 2018 ;
  cc2018_value = Case (cc2018_value = 1 ; "Yes" ; "No" ) ;

  cc2019_field = "Christmas Card 2019" ;
  cc2019_value = C__Company::Christmas Card 2019 ;
  cc2019_value = Case (cc2019_value = 1 ; "Yes" ; "No" ) ;

  emp_field = "Employee Count" ;
  emp_value = C__Company::Employee Count

] ;

JSONFormatElements (
  JSONSetElement ( C__Company::JSON ;
    ["field." & cc2017_field ; cc2017_value ; JSONString ] ;
    ["field." & cc2018_field ; cc2018_value ; JSONString ] ;
    ["field." & cc2019_field ; cc2019_value ; JSONString ] ;
    ["field." & emp_field ; emp_value ; JSONNumber ]
  )))
```

Result:

```
{
  "field" :
  {
    "Christmas Card 2017" : "No",
    "Christmas Card 2018" : "Yes",
    "Christmas Card 2019" : "Yes",
    "Employee Count" : 45
  }
}
```

Quick Best Practice

```
Let ([  
    id_company = Company::ID ;  
    is_active = Company::IsActive ;  
    is_active = case (is_active = 1 , "Yes" , "No")  
];  
JSONSetElement( "{}" ;  
    ["id" ; id_company ; jsonstring] ;  
    ["active" ; is_active ; jsonstring]  
))
```

JSON Field Demo

2.

Preferences (global)
Session Data

Preferences & Session Data

Main Benefits

- ▶ A single global text field in Dashboard or Globals table
- ▶ Extremely easy to access the JSON
- ▶ Replaces numerous \$\$variables
- ▶ Replaces numerous Global Fields in Prefs or Dashboard table
- ▶ Holds structured values, like IDs and field data for a record
- ▶ JSONDeleteElement () useful to clean up a branch

3.

Dashboard Data

Dashboard Data

Main Benefits

- ▶ Uses same Global JSON Field (as defined above)
- ▶ Write values with a script to control when data loads
 - Which is much faster
- ▶ Holds structured values
 - `invoice.rec_count ; invoice.sold ; invoice.due`
- ▶ Holds list of values in JSON Array

Preferences, Session, & Dashboard Data Demo

4.

JSON-LD

For Semantic Search
And Generative AI

JSON-LD & AI

Main Benefits

- ▶ Structured data using schema.org
 - Greatly improves results for both semantic and generative AI
- ▶ Enforces good readability and clear JSON

Sample JSON

```
{  
  "@context": "https://schema.org",  
  "@type": "Person",  
  "name": "Matt Navarre",  
  "job_title": "Trainer",  
  "company_name": "Navarre Training" }  
}
```

Sample FM Code

```
JSONSetElement ( "{}" ;    // Define the JSON-LD type as Note
    ["@type" ; "Note" ; JSONString] ;
    ["data.note_text" ; Note ; JSONString] ;
    ["data.note_date" ; ISO_date (Date) ; JSONString]
)
```

Invoice Line

```
JSONSetElement ( "{}" ;  
    ["@type" ; "Invoice Line Item" ; JSONString] ;  
    ["quantity_ordered" ; Quantity ; JSONNumber] ;  
    ["item_price" ; Price ; JSONNumber] ;  
    ["total_price" ; cTotal ; JSONNumber] ;  
    ["category" ; IL_Product::Category ; JSONString] ;  
    ["brand" ; IL_Product::Brand ; JSONString] ;  
    ["model" ; IL_Product::Model ; JSONString] ;  
    ["color" ; IL_Product::Color ; JSONString] ;  
    ["description" ; IL_Product::Description ; JSONString]  
)
```

Invoice – Uses JSONMakeArray ()

```
JSONSetElement ( "{}" ;  
    ["@type" ; "Invoice" ; JSONString] ;  
    ["invoice_date" ; ISO_date ( INV__Invoice::InvoiceDate ) ; JSONString] ;  
    ["currency" ; "USD" ; JSONString] ;  
    ["invoice_amount" ; INV__Invoice::cAmountInvoiced ; JSONNumber] ;  
    ["paid_amount" ; INV__Invoice::AmountPaid ; JSONNumber] ;  
    ["due_amount" ; INV__Invoice::cAmountDue ; JSONNumber] ;  
    ["items" ; JSONMakeArray (List (Invoice_Lines::cJSON) ; "" ; JSONRaw ) ; JSONArray]  
)
```

Company – Collect it all

```
Let ([ // JSON-LD object with context and type for each record
    person_array = JSONMakeArray (List (C_Person::cJSON) ; "" ; JSONRaw) ; // Must use JSONRaw here
    invoice_array = JSONMakeArray (List (C_Invoice::cJSON) ; "" ; JSONRaw) ;
    notes_array = JSONMakeArray (List (C_Note::cJSON) ; "" ; JSONRaw) ;
    product_interest_array = JSONMakeArray (List (C_Interest::cJSON) ; "" ; JSONRaw) ;
    full_address = C__Company::Address & " , " & C__Company::CityStZip
];
JSONSetElement ("{" ;
    ["@context" ; "https://schema.org" ; JSONString] ; // Add JSON-LD context
    ["@type" ; "Organization" ; JSONString] ; // Define JSON-LD type as Organization
    ["company.name" ; C__Company::Company Name ; JSONString] ;
    ["company.total_invoiced_usd" ; Max (Total Invoiced;0) ; JSONNumber] ;
    ["company.contacts" ; person_array ; JSONArray] ;
    ["company.invoices" ; invoice_array ; JSONArray] ;
    ["company.notes" ; notes_array ; JSONArray] ;
    ["company.product_interest" ; product_interest_array ; JSONArray]))
```

JSON-LD Demo

Thank You!

Contact information:

www.navarre.training

matt@navarre.training

Claris Talk AI on YouTube and podcast platforms