Philipp Puls, 72solutions GmbH, Vienna

Printing in a vertical solution
– –
an introduction to Data-Air-Gaping

Philipp Puls, 72solutions GmbH, Vienna

FileMaker can print, right?

… so what is the problem?

- Layouts are structure driven

  … that is well and fine for the UI

  … that is crap for PrintLayouts

  as they are customer-specific

Consequences

What can go wrong, will

- using the FMDMT for updating would ruin all non standard print-layouts for the customer
- having ALL print-layouts of all customers in the solution seems insecure
- print-layouts are often touched with live data as the form has to follow the function.

Philipp Puls, 72solutions GmbH, Vienna

The classical way

have a print layout file that has all the data files included as External Data Source

+ lightning fast

- breaks, if the structure changes

- every user being able to change the layout has access to all data from the tables included

• ? with layout access you can create a button that calls a script in the included file

The new way

What we wanted

- a user starts a script in a file where he has access to specific data
- this script collects the data that is needed to print the record(s)
- this data is then transferred to the PrintLayouts file
- … tbc

The new way

… continued

- the PrintLayouts file should know as little as possible of the files sending the data
- any Admin should be technically able to change print layouts by himself or by outsourcing to any FM-Consultant
- the transferred data should be efemeral (never hitting the harddrive, not ending up in a backup)

What we did

… to make this happen

- we use the MBS("QuickList") functions as a vehicle to transfer the data across file borders
- we use separate lists for numeric, date and text fields
- we use json data-structure to make the data readable
- we use MagicalValueLists in the PrintLayouts file to cut through the json

What we did

... to make this happen

- We use repeating fields as it is easier to keep 4 calculations in sync than 300!

Problems encountered

… and how we solved them

- we needed to use the FieldID for the DataJSON as node names so renaming a field does not break the layout (as JSONs in FM are alphabetically sorted)
- we split the datatypes in separate QuickLists so you can use proper field-formatting in the layouts.
- …

Problems encountered

... continued

- we need not 3 but at least 6 QuickLists so that we can not only transport the data but also the field-names
- you can reutilise fields, but you can never change the datatype
- if you want to print serverside, you need to name the QuickLists including a UUID so that the print process does not mix the lists.

This is how we do it

We started of with “get a record as JSON” and then we stuff this into the QuickLists

```
//=================
// Autor: 72solutions (PAP)
// 30.04.2020
// Gibt einen Datensatz als JSON
//// Parameter:
//   [x] NormalCalculated
//      Values: "Normal,Calculated" | "Normal" | "Calcualted"
//      defines, what classes of fields should be included
////   [x] includeBinaryData
//      Values: 1 | "" | 0
//      if set to 1, a container Fld will be base64 encoded, otherwise only the name of the
file will be included
//=================
While ([
$_class                 = NormalCalculatedGlobal ;
$_includeBinaryData     = includeBinaryData ;
$_TO        = Case(
                PatternCount(
                    Get ( LayoutTableName );
                "_oB");
                "ALL";
                ""
                ) ;
$_table         = Substitute ( Get ( LayoutTableName ); "_oB"; "");
$_FldList   = ExecuteSQL ( "
                SELECT DISTINCT
                    f.BaseTableName , f.FieldName, f.FieldClass , f.FieldType, f.FieldReps
                FROM
                    FileMaker_BaseTableFields f
                WHERE
                    f.BaseTableName = '"& $_table &"'
                AND
                    '"& $_class &"' LIKE '%'+f.FieldClass+'%'
                AND
                    f.FieldType NOT LIKE 'global%'
               "; "•" ; "¶" )  ;
$_FLDcount  = ValueCount ( $_FldList) ;
$_jsontext  = "";
$_count         = 1
];
$_count         < $_FLDcount + 1
;
[
$_FLDline   = Substitute ( GetValue( $_FldList; $_count ); "•"; "¶");
$_FLDclass  = GetValue ($_FLDline ; 3) ;
$_FLDtype   = GetValue ($_FLDline ; 4) ;
$_FLDreps   = GetAsNumber( GetValue ($_FLDline ; 5)) ;
$_FLDtalbe  = GetValue ($_FLDline ; 1) & Case( $_TO = "ALL";"_oB"; "") ;
$_FLDname   = GetValue ($_FLDline ; 2) ;
$_json      = While ( [
   _FLDclass   = $_FLDclass ;
    _FLDtype    = $_FLDtype ;
    _FLDreps    = $_FLDreps ;
    _FLDtalbe   = $_FLDtalbe ;
    _FLDname    = $_FLDname ;
    _rep        = GetAsNumber(1) ;
    _jsontext   = $_jsontext
   ] ;
   _rep        < _FLDreps + 1
   ;
    [
    _RepString  = Case( _FLDreps > 1; "[" & _rep & "]" ; "");
   _repMinEins = Case( _FLDreps > 1; "[" & _rep - 1 & "]" ; "");

// tbc …
```

```
// … continued
    _filename   = Evaluate
                (
                Case(
                    (PatternCount ( _FLDtype; "binary") and $_includeBinaryData);
                    _FLDtalbe &"::"& _FLDname & _RepString;
                    ""
                    )
                ) ;
   _FLDvalue   = Case(
                (PatternCount ( _FLDtype; "binary") and $_includeBinaryData and
                IsEmpty (Evaluate ( _FLDtalbe &"::"& _FLDname & _RepString)) );
                Substitute(
                JSONSetElement ( "" ;
                    ["FileData" ; "null" ; JSONNull ];
                    ["FileName" ; "null" ; JSONNull ]
                )
                ;["\"{"; "{"];["}\""; "}"] )
                ;
                (PatternCount ( _FLDtype; "binary") and $_includeBinaryData);
                Substitute(
                JSONSetElement ( "" ;
                    ["FileData" ;
                Evaluate (
                "Base64Encode( " & _FLDtalbe & "::" & _FLDname & _RepString
                &")" ) ; JSONString ];
                ["FileName" ; _filename ; JSONString ]
                )
                ;["\"{"; "{"];["}\""; "}"] )
                ;
                Evaluate ( _FLDtalbe &"::"& _FLDname & _RepString)
                ) ;
       _jsonTYPE   = Case(
                IsEmpty ( _FLDvalue );
                    "JSONNull";
                PatternCount ( _FLDtype; "varchar" );
                    "JSONString";
                PatternCount ( _FLDtype; "date" );
                    "JSONString";
                PatternCount ( _FLDtype; "decimal" );
                    "JSONNumber";
                (PatternCount ( _FLDtype; "binary") and $_includeBinaryData);
                    "JSONObject";
                PatternCount ( _FLDtype; "binary" );
                    "JSONString";
                    "JSONString"
                );
        _json   = JSONSetElement (  $_json ;
                    $_FLDid &"•"&_FLDname & _repMinEins ;
                    Case(
                        IsEmpty ( _FLDvalue );
                            _FLDvalue;
                        PatternCount ( _FLDtype; "varchar" ) ;
                            MBS( "Text.Serialize"; _FLDvalue) ;
                        _FLDvalue
                    ) ;
                    Evaluate(_jsonTYPE)
                );
   _rep        = _rep + 1
   ] ;
   _json
   )
;
$_count         = $_count + 1
];

// tbc …
```

```
// … continued
Let(
[
§_json = $_json;
$_json = "";
$_class             = "" ;
$_includeBinaryData = "" ;
$_TO        = "";
$_table         =  "";
$_FldList   =  "";
$_FLDcount  =  "";
$_jsontext  = "";
$_count         =  "";
$_FLDline   =  "";
$_FLDclass  =  "";
$_FLDtype   =  "";
$_FLDreps   =  "";
$_FLDtalbe  =  "";
$_FLDname   =  ""
];
§_json
)
//
)
```

dotfmp.berlin

This is how we do it

Then we realised, we need to get the structure separately, so we can reutilise it:

```
//=================
// Autor: 72solutions (PAP)
// 30.04.2020
// Gibt einen Datensatz als JSON Struktur
//
// Parameter:
//  [x] NormalCalculated
//      Values: "Normal,Calculated" | "Normal" | "Calcualted"
//      defines, what classes of fields should be included
//
//  [x] FeldTyp
//      Values: "varchar,decimal,date" | "Normal" | "Calcualted"
//      defines, what classes of fields should be included//
//=================

Let ([

§_class     = NormalCalculated ;
§_Type      = Lower( VarcharDecimalDate );

§_table         = Substitute ( Get ( LayoutTableName ); "_oB"; "");

§_FldList   = ExecuteSQL ( "
                SELECT DISTINCT
                    f.BaseTableName , f.FieldName, f.FieldClass , f.FieldType, f.FieldReps,f.FieldID
                FROM
                    FileMaker_BaseTableFields f
                WHERE
                    f.BaseTableName = '"& §_table &"'
                AND
                    f.FieldType='" & §_Type & "'
                AND
                    '"& §_class &"' LIKE '%'+f.FieldClass+'%'
                "; "•" ; "¶" ) ;

§_VarName   =  "$$_FldList_" & Substitute ( §_Type & §_class; ","; "" );

§_result    = Evaluate ( "Let ( " & §_VarName   & " = " & Quote( §_FldList)  & " ; 1 ) " )

];

§_result
)
```

This is how we do it

Now we can use this structure given in a global variable to be used on multiple records:

```
//==================
// Autor: 72solutions (PAP)
// 30.04.2020
// Gibt einen Datensatz als JSON Data
//
// Parameter:
//  [x] NormalCalculated
//      Values: "Normal,Calculated" | "Normal" | "Calcualted"
//      defines, what classes of fields should be included
//
//  [x] FeldTyp
//      Values: "varchar,decimal,date" | "Normal" | "Calcualted"
//      defines, what classes of fields should be included//
//==================

While ([


$_FldList   = Evaluate( "$$_FldList_" & Substitute ( VarcharDecimalDate & NormalCalculated; ","; "" )  ) ;

$_TO        = Case(
                PatternCount(
                    Get ( LayoutTableName );
                "_oB");
                "ALL";
                ""
                ) ;

$_FLDcount  = ValueCount ( $_FldList) ;

$_jsontext      = "";

$_count         = 1


];

$_count         < $_FLDcount + 1

;
[
$_FLDline   = Substitute ( GetValue( $_FldList; $_count ); "•"; "¶");

$_FLDclass  = GetValue ($_FLDline ; 3) ;
$_FLDtype   = GetValue ($_FLDline ; 4) ;
$_FLDreps   = GetAsNumber( GetValue ($_FLDline ; 5)) ;
$_FLDtalbe  = GetValue ($_FLDline ; 1) & Case( $_TO = "ALL";"_oB"; "") ;
$_FLDname   = GetValue ($_FLDline ; 2) ;
$_FLDid = Right( "0000" & GetValue ($_FLDline ; 6);4) ;

//  … the rest is the same as in the first CF
```

This is how we do it

On the other side we calculate the field-value in the repeating field as shown here for text-fields:

```
Case(
Extend (VirtualListChild_VirtualList::ID) > MBS( "QuickList.Count"; "Text"& Case (Length ($$uuid); "_"&$$uuid; "") ) ;
"";

Let([
§id         =   Extend (VirtualListChild_VirtualList::ID) - 1;
§ListTyp    =   "Text"& Case (Length ($$uuid); "_"&$$uuid; "");
§rn     =   Get ( CalculationRepetitionNumber );

§json   =   MBS( "QuickList.GetValue"; §ListTyp; §id )
;
§jsonKeys=  JSONListKeys ( MBS( "QuickList.GetValue"; §ListTyp; §id );"")

;

§listCount =    MBS( "QuickList.Count"; §ListTyp )
;
§key = GetValue ( §jsonKeys ; §rn )
;
§Value  =       Case(
                    (§id > §listCount) ;
                        "";
                    Length ( §key )<1 ;
                        "";
                    GetAsText( JSONGetElement ( §json ; §key ))
                )
];

Case( Length( §Value) >0;
MBS( "Text.Deserialize"; §Value );
""
)
)
)
```

Philipp Puls, 72solutions GmbH, Vienna

Let me show you …