NO MORE GLOBAL FILE BUFFERS!

HARM WIBIER

NEW TECHNOLOGY STACK

TECHSTACK?

- With DataFlex 2027 we are introducing a brand-new runtime
  - It will support multiple platforms
    - WebAssembly
    - Linux
    - MacOS
    - Windows
  - It will be used on both the client and the server
    - But we'll also support using the current server technology

BROWSER APPLICATIONS
THE EXPECTED MODEL

- With TechStack we'll have fat clients running the user interface within the browser

- They will only need to access the server for
  - Authentication
  - Data
  - Business processes
  - Reports (might also be done on the client)

- They might store / cache data locally
- They might not need a central server at all

BROWSER APPLICATIONS
THE SERVER

- So, the server needs to expose Web API’s

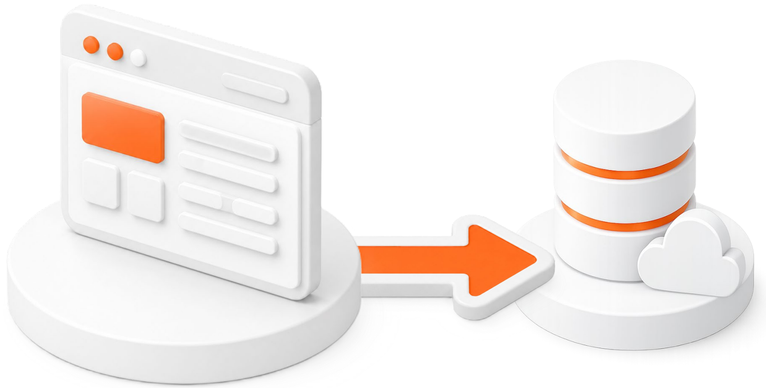
- The server might run on
  - New TechStack runtime
    - With the current WebApp Server / IIS on Windows
    - With the new Web Server on Linux / Docker / Windows / MacOS
  - Current Windows runtime
    - With the current WebApp Server / IIS on Windows

BROWSER APPLICATIONS
THE DATABASE

- Database could be
  - MS SQL
  - Postgress
  - MariaDB
  - MySQL
  - DB2
  - SQLite
  - ..
- On premise, or in the cloud

DESKTOP APPS
THE EXPECTED MODEL

- Fat client running natively on Windows / MacOS / Linux
  - A user interface using FlexTron Technology
- Direct access to the database



DESKTOP APPS
THE EXPECTED MODEL

- Fat client running natively on Windows / MacOS / Linux
  - A user interface using FlexTron Technology
- Direct access to the database
  - *Or maybe it goes through that same Web API*

RATIONALE FOR REDESIGNING THE DATA BINDING

- Everything being inside the windows runtime forces us to rebuild it
- We need to accommodate these new modern deployment scenarios

WHAT PREVENTS FULL COMPATIBILITY?

Objectives

- Deliver data binding and business rule enforcement across both client and server tiers
- Establish more transparent and extensible data connectivity
  - Shift additional logic into DataFlex code for enhanced customization and debugging capabilities
- Implement optimized batch data loading for dataset operations
- Leverage the full capabilities of SQL more effectively
- Maximize the potential of struct-based data handling
- Remove global file buffer

WHY REMOVE THE GLOBAL FILE BUFFER?

- **Transparancy**
  - Difference between global and local record buffer, and when to use which, is always hard to explain to new developers
- **Modularization**
  - Having global buffers accessed throughout the entire application is simply not modular
- **Stateless**
  - To be able to send operations to a Web API they need to be stateless
  - Global record buffer in DataFlex represented the state of the driver
- **Optimization**
  - No more copying data back and forth between local and global file buffers

WE DO LIKE THE SYNTAX

1

```
Showln Customer.Name
```

2

```
String sVal
Get Field_Current_Value of oCustomerDD Field Customer.Name to sVal
Showln sVal
```

3

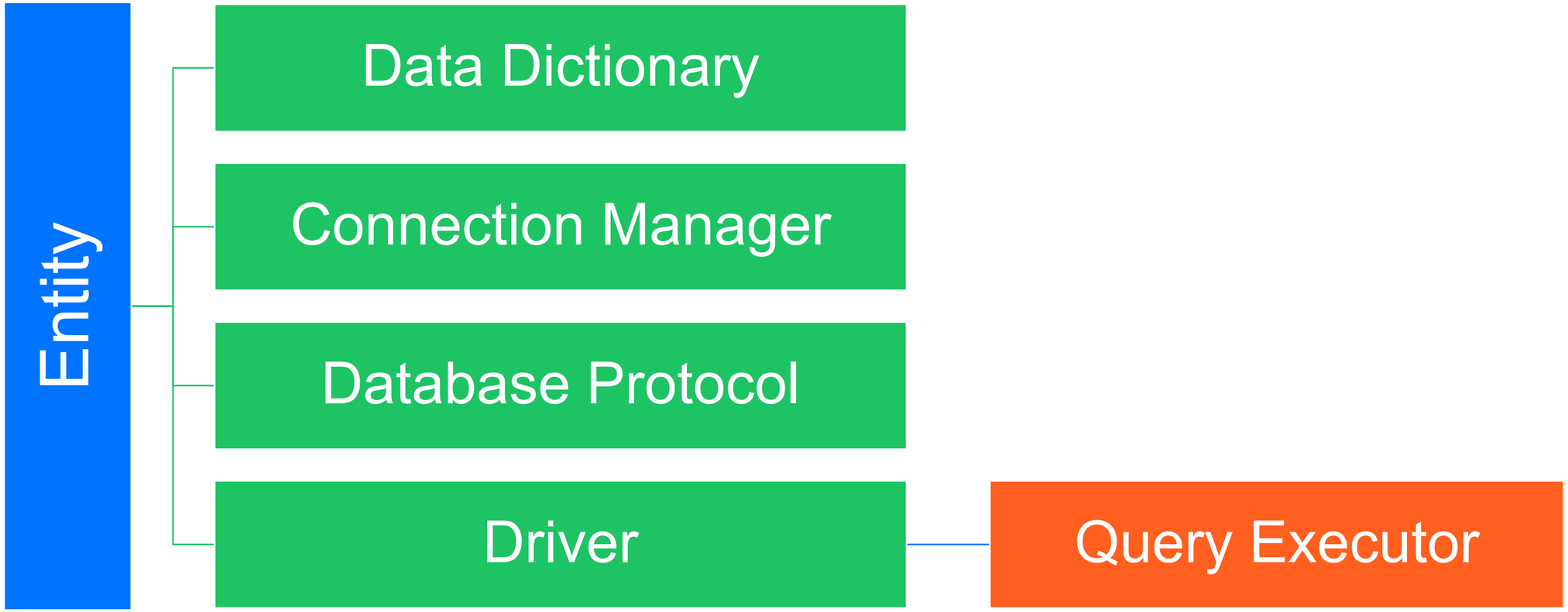
```
Showln (Field_Current_Value(oCustomerDD, RefTable(Customer.Name)))
```

THE PLAN!

THE BATTLE PLAN

- Describe tables in DataFlex code using struct-like syntax
- Use structs for record buffers and for transporting data
- Make data dictionaries responsible for initiating database operations
  - Support convenient syntax of global buffers on the local buffers
- Design stateless message protocol for driver communication
  - Have drivers that communicate with HTTP API’s
  - Have drivers that generate SQL and leverage JOIN’s to fetch parents in one go
- Write it all in DataFlex code
  - Debuggability and extensibility

COMPONENTS


Entity
Data Dictionary
Connection Manager
Database Protocol
Driver
Query Executor

DATA BINDING
ENTITIES

- Declare the data model directly within DataFlex source code
  - Enhances maintainability and streamlines version control
- Provides metadata to all the layers
- Defines a struct that
  - Serves as the record buffer
  - Facilitates data transport across layers

```
{ Connection="OrderData" }
Entity Inventory
    {
        PrimaryKey=On
    }
    String Item_ID
    String Description
    Integer Vendor_ID
    String Vendor_Part_ID
    Number Unit_Price
    Integer On_Hand

    Add_Relation Vendor_ID to Vendor.ID

    { Unique=On }
    Add_Index Item_ID ASC
    { Unique=On }
    Add_Index Vendor_ID ASC Vendor_Part_ID ASC Item_ID ASC
    { Unique=On }
    Add_Index Description ASC Item_ID ASC
End_Entity
```

DATA BINDING
DATA DICTIONARIES

- Each Data Dictionary is associated with a specific entity
- They serve as the central access point for all database operations
  - This eliminates the distinction between global and local file buffers
- Now implemented for 97% in DataFlex code
  - The base class cEntity resides in the runtime
    - Exposes entity metadata for programmatic access
    - Supplies a single entity instance that acts as the record buffer
  - The compiler and runtime enable field access via <object>.<field> syntax
- Commands are now fully compatible with Data Dictionary objects
- Communicates with drivers through stateless messaging

```
Use Db\cDataDictionary.pkg

{ Connection="OrderData" }
Entity SalesPerson
    { PrimaryKey=On }
    String ID
    String Name
    String Avatar

    Add_Index ID ASC
    Add_Index Name ASC ID ASC
End_Entity

{ Entity=SalesPerson }
Class cSalesPersonDataDictionary is a cDataDictionary
    Procedure Construct_Object
        Forward Send Construct_Object

        Set phEntity to (RefEntity(SalesPerson))

        Set Foreign_Field_Option DD_KEYFIELD DD_NOPUT to True

        Set Field_Label_Long (RefTable(Self.Id)) to "Sales Person ID"
        Set Field_Label_Short (RefTable(Self.Id)) to "ID"
        Set Field_Option (RefTable(Self.Id)) DD_REQUIRED to True
    End_Procedure
End_Class
```

/*50 YEARS*/

DATA BINDING
DIRECT FIELD ACCESS

- oDataDictionaryObject.FieldName
  - Allows direct access to field values
  - Allows convenient syntax known from global file buffers
  - Provides migration path for code using global file buffers
- Self.FieldName can be used in DD class code

- Used with
  - Find
  - Expressions
  - Move
  - Entry_Item
  - RefTable

```
Find GT oInventoryDD.Item_ID

If (oCustomerDD.Status = "Y") Begin

Move "Book Store" to oCustomerDD.Name

Entry_Item oCustomerDD.EMail_Address

Set Field_Option (RefTable(Self.Name)) DD_REQUIRED to True
```

- Demo…

DATA BINDING
ALIAS FILES

- Name meta tag on entity allows different database name
- Queries will be generated with ‘AS ..’ entity name to allow native expressions to work

```
{ Name="Employee" }
Entity Manager
    {
        PrimaryKey=On
        AutoIncrement=On
    }
    Integer EmployeeId
    String Name
    Integer ManagerId

    Add_Index EmployeeId ASC
End_Entity
```

DATA BINDING
DRIVER API

- Implemented as DataFlex classes and objects
  - Optional native code via External_Class and External_Function
- Stateless API defined as message structs
  - Records passed as Entity structs or Variant arrays
  - Leverages the new struct reflection APIs
  - Messages: Find, Create, Update, Delete, Query, CreateDb, Exists
- Spec to be public, documented, and versioned at stabilization
  - Supports custom driver implementations

DATA BINDING
STANDARD DRIVERS

- Will do query generation in DataFlex code
  - String array as string builder with new StringJoinFromArray runtime function and internal refcounted string handling makes this perform well
- Only a thin layer in native code exposing itself as External_Class to DataFlex

DATA BINDING
SQLITE

- First driver available
- Available on all platforms including WebAssembly

- Quite comprehensive SQL support
- Stores data in a single file on disk (single .db file containing multiple tables)

- Expected to replace Embedded Database in samples due to its portability

https://www.sqlite.org/

DATA BINDING
SQL DRIVER

- Based on SQLAPI++
- Planned support for
  - MS SQL
  - PostgreSQL
  - MySQL
  - MariaDB
  - DB2
  - ODBC
- Will be available on Native platforms (Windows, Linux, MacOS)
  - Not on WebAssembly

DATA BINDING
DIFFERRENCES WITH CURRENT WINDOWS APPROACH

- Drivers assume the database structure to be correct
  - No more querying of the structure on startup
  - API’s for database creation and alteration will be provided
- Much more code driven
  - No more filelist.cfg
  - No more .int files
- Indexes are ‘optional’
  - Depending on the driver

DATA BINDING
CONNECTIONS

- Connections are DataFlex Objects
  - Data Dictionary Objects can be configured manually to use a specific object
    - This will override configured managed connections
- Connection Manager available for managing connections
  - Meta Tag on Entity configures the connection id
  - Data Dictionary will query connection manager for connection
- URI based connection strings

```
<scheme>://<user>:<password>@<host>/<path>?<option>=<val>

sqlite:OrderEntryFull.db?mode=rwc
sqlserver://Harm:Test1234@HW-LAPTOP\SQLEXPRESS/AdventureWorks2025?trust_server_certificate=true
```

DATA BINDING
CONNECTION SAMPLE 1

- Automatic instantiation of driver object instance

```
{ Connection="OrderData" }
Entity Vendor
// ...

Object oConnectionManager is a cDbConnectionManager
    Send RegisterDriver "sqlite" (RefClass(cDbSqliteDriver))
    Send RegisterConnection "OrderData" "sqlite:OrderEntryFull.db?mode=rwc"
End_Object
```

DATA BINDING
CONNECTION SAMPLE 2

- Manual driver object instantiation (driver registers automatically)

```
{ Connection="OrderData" }
Entity Vendor
// ...

Object oConnectionManager is a cDbConnectionManager
    Object oSqliteConnection is a cDbSqliteDriver
        Set psConnectionId to "OrderData"
        Set psConnectionUri to "sqlite:OrderEntryFull.db?mode=rwc"
    End_Object
End_Object
```

DATA BINDING IN DEPTH
CONSTRAINTS

- Constrain command works with the new Data Dictionaries
- Constraints are stored in memory (struct properties) and are passed to the driver
- Driver is responsible for applying contraints and the SQL drivers will generate a WHERE clause
- Expression constraints are not supported

- This gives us the performance of SQL filters with the advantages of constraints
  - Backend independent syntax
  - Compile time syntax checking
  - Code completion

DATA BINDING
NATIVE QUERY SUPPORT

```
Constrain oCustomerDD as @SQL”Customer.Name LIKE ‘%Miles%’”
```

- Planned support (not yet implemented)

```
Entry_Item @SQL"CONCAT(Presenter.FirstName, ' ', Presenter.LastName)" from oPresenterDD
```

- Limited support, only available on cWebColumn and cDbQuery right now

DATA BINDING
USING ENTITY AS STRUCT

- Capability that is used within data dictionaries and the drivers
  - Data Dictionaries keep a struct instance as record buffer
  - Drivers return records in struct instances
- Structs being passed around in the runtime as refcounted pointers, which are only copied on write provides optimal performance

- Property Variant **Entity_Value**
  - Direct access to the record buffer struct, updating it does not affect change states
- Procedure **UpdateAllFields** Variant vRecord
  - Updates all field values and sets their changed-state

DATA BINDING IN DEPTH
DBQUERY

- New API for loading multiple records
- Only loads specified columns
- Supports relations
- Supports constraints
- New cDbQuery class
- Supports paged data loading
  - X number of records from record with specified RowId
- Used by cWebList by default

```
Object oMyQuery is a cDbQuery
    Set Server to oOrderDetailDD

    Constrain oCustomerDD.Customer_Number eq 46

    Entry_Item oOrderDetailDD.Order_Number
    Entry_Item oOrderDetailDD.Qty_Ordered
    Entry_Item oOrderDetailDD.Price
    Entry_Item oInventoryDD.Item_ID
    Entry_Item oCustomerDD.Name
End_Object

Procedure RunQuery
    Variant[][] aRecords
    Variant[] aRowIds
    Integer iItem iItemTo

    Get FetchFirst of oMyQuery 1000 (&aRowIds) to aRecords

    Move (SizeOfArray(aRecords) - 1) to iItemTo
    For iItem from 0 to iItemTo
        Showln aRecords[iItem][0] " " aRecords[iItem][1] " " aRecords[iItem][2] " " aRecor
    Loop
End_Procedure
```

/*50 YEARS*/

DATA BINDING
DDP – DATA DICTIONARY PROTOCOL

- Do I always have to synchronize data or use the HttpClient to upload/download data? In a future tech stack version, no.
- If your client and server share the entity model DDP can directly query/store/update records over HTTP.

DATA BINDING
DDP – DATA DICTIONARY PROTOCOOL

- It’s a module for the WebAPI framework and as such currently works for SQL-based DataFlex Windows deployments
- Why Current-Gen first? To provide a migration path
  - Expose your DataSet on a DataFlex Windows –based WebApp
  - Create a client app for mobile to run besides your production webapp
- Best performance, stability will be provided with a 1 to 1 TS server and client
- Demo of querying

```
//  With the cWebHttpHandler you handle complete HTTP requests.
Object oRestFramework is a cWebApi
    //  The psPath property determines the path in the URL for whic
    Set psPath to "Api"

    Send AddIterator (RefClass(cJSONIterator)) "application/json"
    Send AddIterator (RefClass(cXMLIterator)) "application/xml"

    Use ApiCallLogger.pkg

    Object oDDPAPI is a cWebAPIDataDictionaryProtocol
        Set psPath to "ddp"

        Send Expose (RefClass(cCustomerDataDictionary))
        Send Expose (RefClass(cSalesPersonDataDictionary))
        Send Expose (RefClass(cOrderHeaderDataDictionary))
        Send Expose (RefClass(cVendorDataDictionary))
        Send Expose (RefClass(cInventoryDataDictionary))
        Send Expose (RefClass(cOrderDetailDataDictionary))
    End_Object
```

BUSINESS RULES

DATA BINDING IN DEPTH
META DATA

```
{ Connection="OrderData" }
Entity OrderHeader
    {
        PrimaryKey=On
        AutoIncrement=On
    }
    Integer Order_Number
```

- Each entity carries its own static meta data
  - Defined with meta tags right in the source
  - Works at the entity, field, index, or relation level
  - Read-only access through the cEntity class, which is the base class of cDataDictionary
  - Mainly aimed at drivers
- The Data Dictionary adds its own meta data layer on top
  - Name/value pairs managed per DD and per field
  - Falls back to the entity values by default
  - Forms the foundation for validations and business rules

DATA BINDING IN DEPTH
EVENT DRIVEN BUSINESS RULES ENGINE

- Subscribe to events for DD operations
  - OnValidate
  - OnFieldValidate
  - OnFieldValueChanged
  - …
- Subscribe to events for implementing field options
  - Central administration of field attribute event handlers
- Supports subclass / mixin pattern but also separate stateless validator objects

```
Use System\Globals.pkg
Use Db\cDD_Save_mixin.pkg
Use Db\cDD_MetaData_mixin.pkg

Define DD_Capslock for "CAPSLOCK"

Class cDD_Capslock_mixin is a Mixin
    Procedure Adjust_DD_Capslock tDD_OnFieldValueChange ByRef details Boolean ByRef bCancel
        Move (Uppercase(details.vValue)) to details.vValue
    End_Procedure


    Procedure Set_DD_Capslock tDD_OnSetFieldAttribute ByRef details Boolean ByRef bCancel
        If (DD_MetaValueIsTrue(details.vValue)) Begin
            Send OnFieldValueChange_Add details.hField (RefProc(Adjust_DD_Capslock)) Self
        End
        Else Begin
            Send OnFieldValueChange_Del details.hField (RefProc(Adjust_DD_Capslock)) Self
        End
        Move True to details.bStored
    End_Procedure

    Procedure Get_DD_Capslock tDD_OnGetFieldAttribute ByRef details Boolean ByRef bCancel
        Move (OnFieldValueChange_Has(Self, details.hField, (RefProc(Adjust_DD_Capslock)), Self)) to details.vValue
    End_Procedure
End_Class

Send OnSetFieldAttribute_Add of ghoGlobalDDMetaDataHandlers DD_Capslock (RefProc(Set_DD_Capslock)) C_UnresolvedObj
Send OnGetFieldAttribute_Add of ghoGlobalDDMetaDataHandlers DD_Capslock (RefProc(Get_DD_Capslock)) C_UnresolvedObj
                                                                                                   /*50 YEARS*/
```

```
Use Db\cDataDictionary.pkg
Use Db\cDD_Field_Validator.pkg

Define DD_BasicEmail for "EMAIL-BASIC"

Object oBasicEmailValidator is a cDD_Field_Validator
    Set psOptionId to DD_BasicEmail

    Procedure Validate_DD_Field tDD_OnFieldValidate ByRef details Boolean ByRef bCancel
        Integer iAt iDot
        String sValue

        Get Field_Current_Value of details.hoDD details.hField to sValue
        If (Trim(sValue) <> "") Begin
            Move (Pos("@", sValue)) to iAt
            Move (RightPos(".", sValue)) to iDot

            If (not(iAt > 1 and iDot > 0 and iDot > iAt + 1 and iDot < Length(sValue))) Begin
                Send FieldError of details.hoDD details.hField DFERR_OPERATOR "Please enter a valid em
                Move True to details.bInvalid
            End
        End
    End_Procedure
End_Object
```

/*50 YEARS*/

FLEXIBLE AND TRANSPARENT APPROACH THAT CARRIES THE DATAFLEX DNA FORWARD

DATAFLEX 27.0 - TECHNOLOGY PREVIEW

- Coming soon!

- Enough to show and proof the data binding concept
  - Functioning SQLite driver
  - SQL Driver and DDP will be added later