# Understanding the Architecture of the New Grids

John Tuohy

# The New Grids

- The new grids classes were created to replace the functionality of the existing grid classes
  - We needed to provide replacement functionality for:
    - Regular grids (the old Grid class)
    - Data Aware Entry grids (the old dbGrid class)
    - Prompt lists (the old dbList and List class)
- A new grid is implemented by using the Codejock COM ReportControl
- It is modeled using a set of composite classes that represents
  - The grid
  - The columns
  - The data (i.e., the data-source)

# The New Grid Class Structure

cCJComReportControl
--- cCJGrid
------ cCJGridPromptList
------ cDbCJGrid
--------- cDbCJGridPromptList

cCJComReportColumn
--- cCJGridColumn
------ cDbCJGridColumn

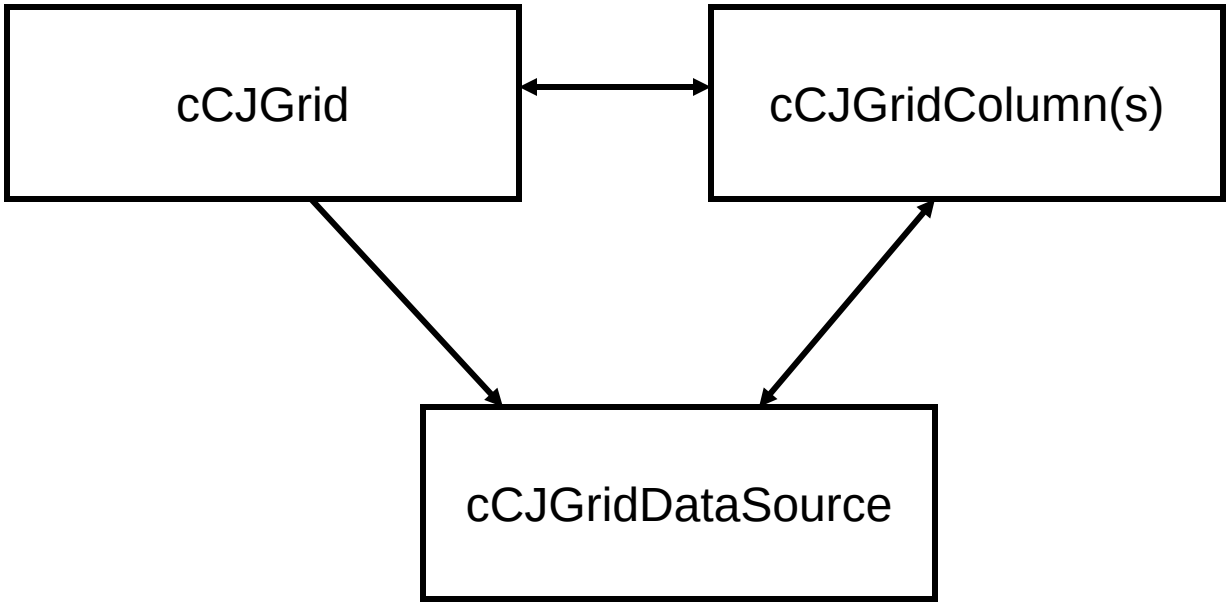
cObject
--- cCJGridDataSource
------ cCJGridCachedDataSource
--------- cDbCJGridDataSource

# The Composite Classes

- The Interface
  - All grid, column and data-source objects communicate through a standard abstract interface
  - This interface contract must be upheld
  - The interface and even the direction of the interface was designed to be as simple as possible. For example:
    - The data source object sends no messages to grid objects and only sends a few messages to the grid column object
    - The grid object has no idea how data is maintained in the data source
    - The grid object knows nothing about column meta-data (data type, mask, appearance)
  - The interface can be extended in sub-class sets
    - An extension requires sub-classing all objects
    - An attempt was made to keep the need for extensions minimal
- The Implementation
  - The actual implementation of these interfaces is up to the class
  - As long as the interface is supported the implementation can be anything
  - We provide those implementations in our classes
  - You may replace or augment these implementations in your classes

# Regular Grids - cCJGrid

## Grid Composite Object Structure



# Grid Composite Object Structure

cCJGrid or cCJGridPromptList

(*cCJGridDataSource*)

cCJGridColumn Object.1
:
cCJGridColumn Object.n

# A cCJGrid Object

```
Object oGrid is a cCJGrid
    Set Size to 230 408
    Set Location to 7 9

    Object oName is a cCJGridColumn
        Set psCaption to "Name"
        Set piWidth to 95
    End_Object

    Object oType is a cCJGridColumn
        Set psCaption to "Type"
        Set piWidth to 55
    End_Object

    Object oSize is a cCJGridColumn
        Set psCaption to "Size"
        Set piWidth to 50
    End_Object

End_Object
```

# The Grid Class

- The grid class controls the overall behavior of the grid
  - It handles things like overall appearance, edit modes, selection modes, activation, navigation, saves and deletes
  - Most external events and messages are sent to the grid where it either:
    - Handles the event or message
    - Sends it to the appropriate column object
    - Sends it to the data source object
  - It has a lot of properties
    - This is where we expose the wealth of grid attributes provided by the Codejock control

# The Grid Column Class

- Much of the code that controls how a grid behaves will reside in the grid column objects and not in the grid object
- Column properties determine a column's appearance and behavior
- Most custom events for navigation and validation will be placed in grid column objects

# The Grid Column Class

- When designing your grid remember that columns can be moved around and hidden at runtime
  - Don't assume that you know what the order of navigation is
  - When accessing other grid columns it is best to access them by their object name (which doesn't care about order)
- Each column is assigned an item order, which is determined by the creation order of your grid columns
  - This order determines the order of columns values in your data source array
  - The piColumnId property returns a column's creation order
  - This can and should be used when filling data into a data source

# Using piColumnId

- This uses piColumnId to load column values into the data source array. This is more robust than using the ordinal values (0, 1 and 2). Assume there are two grid columns oColumnChoice and oColumnId

```
Procedure LoadMyData
    tDataSourceRow[] DataSourceArray
    Integer iChoice iId
    Get ColumnId of oColumnChoice to iChoice
    Get ColumnId of oColumnId to iId
    Move "First Choice"  to DataSourceArray[0].sValue[iChoice]
    Move "A"             to DataSourceArray[0].sValue[iId]
    Move "Second Choice" to DataSourceArray[1].sValue[iChoice]
    Move "B"             to DataSourceArray[1].sValue[iId]
    Move "Third Choice"  to DataSourceArray[2].sValue[iChoice]
    Move "C"             to DataSourceArray[2].sValue[iId]
    Send InitializeData DataSourceArray
End_Procedure
```

# The Data Source Class

- The data source manages the data
- The grid and grid column objects treat the datasource as a black box
  - When they need data they ask for it
  - When they need to update the data for a column in the current row, the column object tells the data source to update a value
- Row edits, inserts and deletions are handled by the data source
  - The data source will keep itself up to date with changes
  - If needed the data source will update external sources (DDs) with changes
- The data source is the silent partner in the grid and you will have limited interactions with it:
  - The object is created automatically so you won't see it in your code
  - There are few properties to set
  - The data source keeps track of the current row (SelectedRow)
  - With a static grid you will send messages to process data
  - With a dynamic data aware grid you may never send a message to it
- While silent, it is a very sophisticated class

# cCJGrid

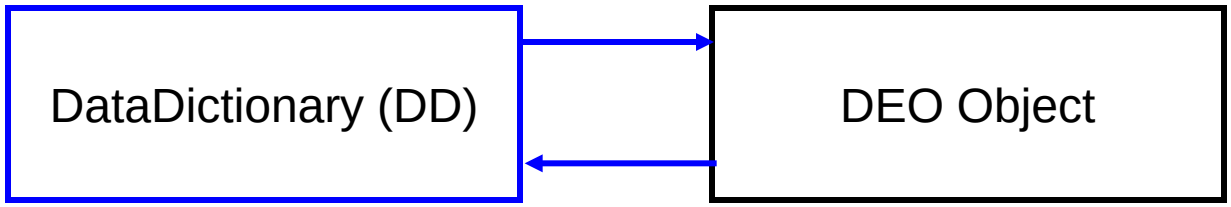
- Let’s see some examples…

# The Data Aware Grids

- The data aware classes are sub-classes of the three grid composite classes and turns the grid into a data-entry object (DEO)
- cDbCJGrid
  - is extended so that it understands and implements the Data Dictionary to DEO interface contract
- cDbCJGridColumn
  - is extended so it can be bound to DD objects using the Entry_Item command
- cDbCJGridDataSource
  - is extended so that it is always synchronized with a DD server
  - is extended to support a dynamic data source allowing for partial loading of data, and caching of data

# Data Aware cDbCJGrids

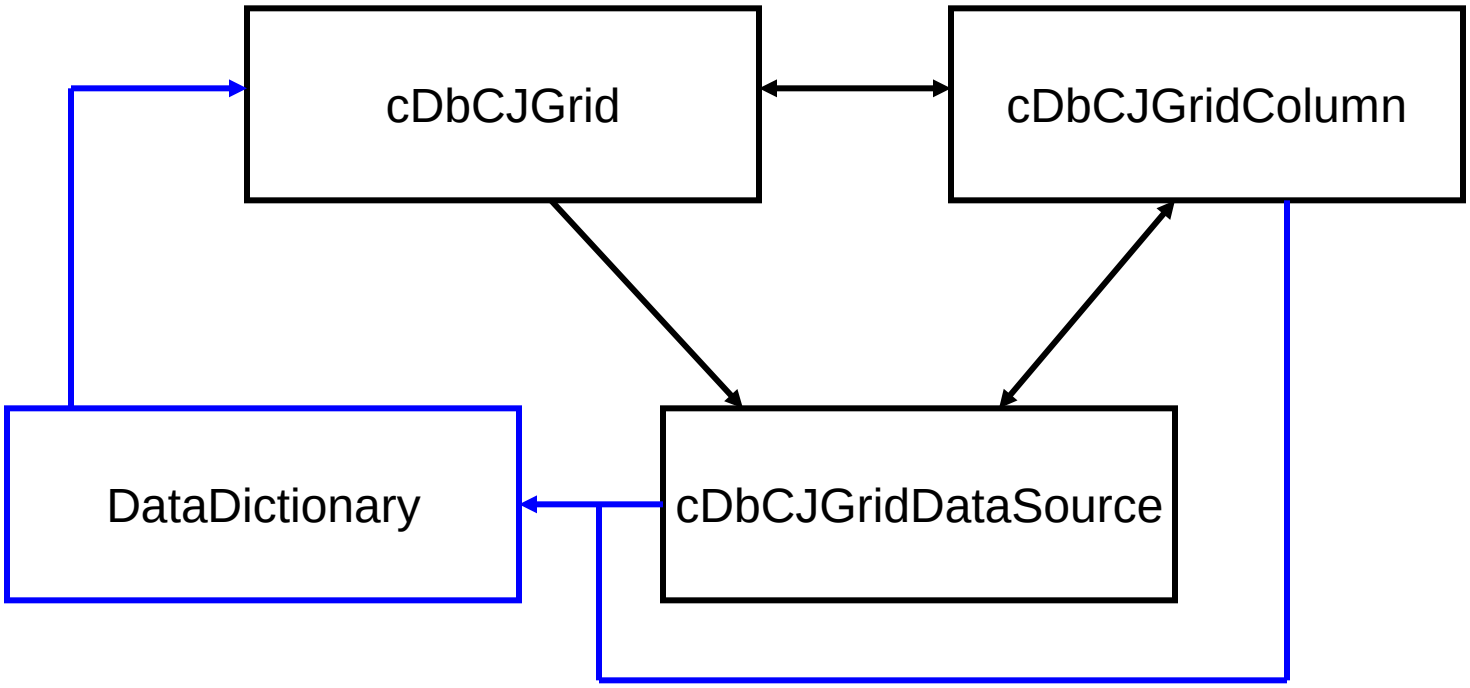
## The DD / DEO interface contract




- DDs and DEOs communicate with each other via an interface contract
- DDs keep track of all client DEOs they serve
  - Any change in a DD will send messages to its client DEOs
- A DEO knows which DD server it is attached to
  - A DEO uses its DD to determine how it should behave
  - A change in a DEO will sends messages to its server DD
  - A DEO will send request messages to its server DD

# Data Aware cDbCJGrids

## Data Aware Grid Object Structure



# Db-Grid Composite Object Structure

cDbCJGrid or cDbCJGridPromptList

(*cDbCJGridDataSource*)

cDbCJGridColumn Object.1
:
cDbCJGridColumn Object.n

# A cDbCJGrid Object

```
Object oCustGrid is a cDbCJGrid
    Set Size to 180 281
    Set Location to 6 6
    Set Ordering to 1
    Set Server to Customer_DD

    Object oNumCol is a cDbCJGridColumn
        Entry_Item Customer.Customer_Number
        Set piWidth to 38
        Set psCaption to "Number"
    End_Object

    Object oNameCol is a cDbCJGridColumn
        Entry_Item Customer.Name
        Set piWidth to 183
        Set psCaption to "Customer Name"
    End_Object

    Object oEmail is a cDbCJGridColumn
        Entry_Item Customer.EMail_Address
        Set piWidth to 50
        Set psCaption to "email"
    End_Object

End_Object
```

# Static and dynamic data sources

- A static grid loads all data at once.
  - It is your responsibility to load the data into the grid*
  - It is your responsibility to process your grid data*

  * Usually (more in next page)
- A dynamic grid loads and caches data as needed
  - Usually dynamic data is loaded and maintained automatically
- Regular grids (cCJGrid, cCJGridPromptList) are static
- Data aware grids (cDbGrid, cDbCJGridPromptList) are dynamic by default but can be set to be static

# The dynamic data source

- Data aware grids (cDbGrid, cDbCJGridPromptList) are by default dynamic
  - The data source loads, caches and refreshes data as needed
  - When rows are saved, the data source updates the DDs and the tables
- Data aware grids can be set to be static
  - This powerful feature is controlled by one property - *pbStaticData*
  - The data can be loaded manually or automatically.
  - If the data is not yet loaded, the first time data is needed, it just loads all of the data
  - Static data-aware grids still refreshes the current row with the latest DD data
  - Saving a row in a static grid does a full DD Save
  - Clearing the data source will cause the grid to refresh itself the next time it needs data

# Use the cCJGrid or the cDbCJGrid?

- The non-data aware cCJGrid is much more powerful and full featured than our old grid class
  - You may find that many of your more advanced grid needs can be handled by this class
  - Customizing the cCJGrid class is easier that customizing the data-aware cDbCJGrid class.
  - When in doubt, start with the cCJGrid class
    - In the long run it is usually easier to take a simpler class and add capabilities than to take a more complex class and remove capabilities

# The current row

- The grid operates around the concept of a current row (*SelectedRow*)
  - The row you have navigated to and possibly editing is the SelectedRow
- Row data values can be accessed through the column object
  - *SelectedRowValue* accesses the current row
  - *RowValue* accesses any row
- You should only change values in the current row
  - Those values changed via the column object's *UpdateCurrentValue* method
  - The current row concept allows the grid to utilize the DataFlex row/record logic (edit, validate, verify, etc.)
- The data source can be directly changed using a batch mode

# Multi-Select Lists

- Grids can be no-select, single-select or multi-Select
  - This is all controlled by various grid properties
  - Selections can be made via standard mouse keyboard events or programmatically
  - Various methods allows you to process selected items
- The current row (*SelectedRow*) is different than a selected row
  - sorry about the name
- Multi-select should only be used with static grids

# Changing multiple Values

- Batch changes to a grid, where you change a number of rows at one time should be done by operating directly on the data source.
  - Load the data source array (*Get DataSource*)
  - Change whatever you want directly in the array
  - Put the data back into the grid using *InitializeData* or *ReInitializeData*
- This is *much* more efficient than trying to save and change each row within the grid
- This bypasses the normal save logic (validation, verification, etc.), which is probably what you want
- You would only do this with static grids

# The legacy grid classes

- New Grids versus Old Grids
  - The new grids are not meant to be interface compatible with the old grids
  - The new grids are not meant to be feature equivalent with the old grids
    - Although we added a lot more equivalence than we planned
  - The new grids are meant to be interface and feature compatible with the DataFlex Framework and with DDs
  - The old grids still work
    - They can be used side by side with the new grids
    - There will be few changes in these old legacy classes

# The Codejock Report Control

- The public interface consists of the COM classes and methods and the higher level COM classes and methods we've built on top
  - The COM interfaces are prefixes with *cCJCom*, *Com* and *OnCom*
- Our interface isolates you from the COM interface level
  - Always start by using our interface
  - If required the entire COM interface is available for customization
    - Recognize that working at the COM level is working at a lower level. This requires more knowledge and more time
- This is designed to be a Grid and not a Report Control
    - You can use this COM control as a Report Control
    - If you are looking for a *great* reporting control class check out the "VDF SIG Codejock Library"

# The Data Source and Virtual Mode

- The report control supports a feature called “Virtual Mode”
  - When enabled, the grids data is stored in a an external source that the grid knows knows nothing about
  - The grid only knows how many rows it has, which is used to position the scrollbar thumb
  - When the grid needs to paint a cell, it sends an event asking for data for a row and cell value.
  - It is up to this event to provide a value
  - This event can be called for any cell at any time
  - This event gets called quite often. It has to be fast
- Our Grid class uses virtual mode to separate the grid from the data
  - We use the cell paint event to provide the grid with requested cell data
  - When data is edited, we update the data source which indirectly updates what you see
  - This allows us to support a dynamic data source

# Report versus Grid Control

- The Grid
  - Is designed for row oriented data-entry
  - It uses the DataFlex model for navigation, edit, validations and verifications
  - It can work with dynamic data. Data is acquired from a data source object supporting caching, partial loading of data
  - Is in “grid” format – its just cells of rows and columns. No fancy nesting or grouping
- The Report Control
  - Is designed as a reporting control
  - All data is loaded directly into the grid. No caching.
  - Has great native sorting, grouping and filtering capabilities
  - You are working at a lower level.
  - Has very limited data entry capabilities
- Which Control to Use
  - Usually the choice is clear
  - There are cases where either control would do the job
    - A static read-only multi-select list would work with either
  - If you are unsure, use the Grid