iConnect 188325 Data Grid Design Approach

Project Overview

Wizard-based user interfaces often require editable and non-editable data to be displayed in a tabular format. Within FireLight, we are giving users the ability to use the designer to create a grid structure that is bound to a custom list. The grid can contain any combination of columns with check boxes, radio buttons, text boxes, and simple display text.

Features/Requirements

- The user can add a new data grid to a wizard.
- A custom list must be selected as a backing data source for the grid.
- The data in the custom list will be used to pre-populate the values in the grid. The user can specify in the designer what can be done with the data in the columns. Available column types are Key (uniquely identifies each row of data), Text (display only), Text Box (editable text), Check Box (Multiple Selection), and Radio Button (single selection).
- Rules can be used to change data within the grid, change the custom list to which the grid is bound, and extract data from individual cells within the grid.
- If the custom list that is used in a grid within an existing activity is changed, either within the admin or by a rule, the data within the grid will be re-defaulted the next time the activity is loaded.

Use Cases / Workflow Changes

- Fund Allocation
- Contingent Beneficiary / Other Insured Lists

Admin Changes

- The data grid has been added to the wizard designer's toolbox to allow users to drag and drop a new data grid on a page.
- Field properties specific to data grids were added.

App Changes

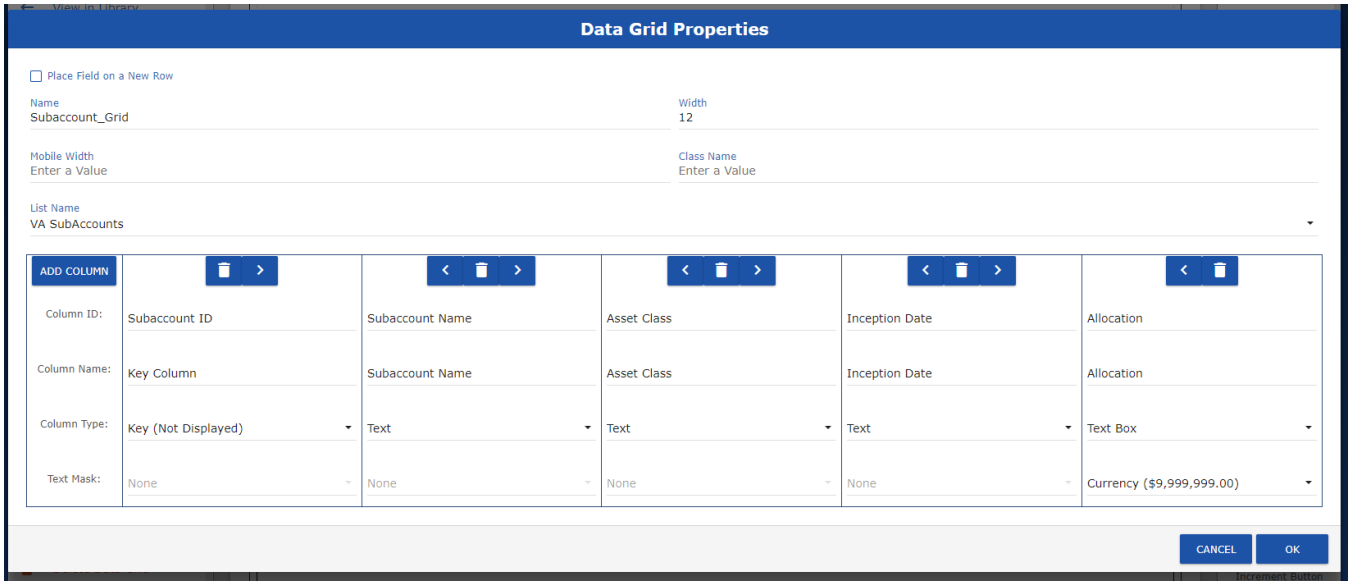
- The wizard generator has been modified to handle rendering of the data grid component.
- Additional rule nodes have been added to the rules engine to accommodate data grids.

Integration Changes

No changes.

UI Mock Ups

Admin – Field Properties



App – Rendered Data Grid

Subaccount Name	Asset Class	Inception Date	Allocation
Fixed Account	N/A	Sep-06-2018	$100
Money Market	Cash Equivalents	Jun-15-2002	$0

How to Enable and Use This Feature

Example #1 – Subaccount Allocation Grid

1. Add a new custom list in the FireLight Admin that contains the information for the subaccounts. The list should contain five columns: Subaccount ID, Subaccount Name, Asset Class, Inception Date, and Allocation. The information in the custom list will be used when initially populating the data source for the grid.
2. In the wizard designer, drag a new Data Grid from the toolbox on the right-hand side of the screen into the designer surface.
3. Double-click on the grid to open up the grid properties dialog.
4. In the Name field, give the grid a name that is easy to remember, since this will be used later when writing rules.
5. Click on the List Name dropdown and select the name of the list that was uploaded in step 1.
6. Initially, the grid will have a single column. Add four more by clicking the Add Column button.

7. Enter a Column ID and Name for each column. The Column ID should match up with the name of the column in the list that was uploaded in step 1. The name will be what the user sees as the header caption when the grid is rendered.
8. Select Column Type for each column. The columns and their functions are listed below. For the example above, the first row should be a Key column, the last column should be a Text Box column, and all other columns should be Text.
    a. **Key (Not Displayed)** – A column will be created in the data source for the grid, but the column will not be visible. This is useful for providing a unique identifier for each row of data.
    b. **Text** – A column of display-only data will be created.
    c. **Text Box** – A column of editable data will be created. Optionally, a Text Mask can be specified for formatting the data in the text box. The Text Mask selection will only be visible if one of the columns has been assigned as a Text Box.
    d. **Check Box** – A column of selectable data will be created. Multiple row selections are allowed.
    e. **Radio Button** – A column of selectable data will be created. Only a single row can be selected.
9. Click the OK button to close the dialog and then click the Save Wizard button to save the wizard.
10. At this point, no further changes are necessary. The grid will be bound to the custom list when a new activity is created in the app.

Example #2 – Fund Realignment

1. Fund realignments can be accomplished by creating a new custom list with the new funds and then changing the grid's list, either in the designer or via a rule. FireLight does not have the ability to map allocations from one fund to another, so all data in an existing activity's grid will be cleared if the backing custom list is changed.
2. In this case, we will use a rule to ensure that the new fund list is loaded if the activity is loaded after midnight UTC on 1/1/2018. The rule is below.

```
<table type="Grid" list="VA SubAccounts" name="SubGrid"
dataitemid="SubAccount_Grid">
  <propset name="ListName" prop="Value">
    <get name="SubGrid" />
    <conditional>
      <compare op="&gt;">
        <datetimedifference option="Second">
          <datetimeutcnow />
          <const value="1/1/2018" />
        </datetimedifference>
        <const value="0" />
      </compare>
      <const value="VA SubAccounts 2" />
      <const value="VA SubAccounts" />
    </conditional>
  </propset>
</table>
```

Example #3 – Validation

1. Validation makes use of the cellget, and cellset rule nodes. In the example below, assume we have a subaccount grid with the ability for the user to enter specific allocation percentages for the various funds. In that case, we want to validate each row to ensure that the user has not entered an allocation greater than 100%.

```
<table type="Grid" name="SubGrid" dataitemid="SubAccount_Grid">
  <foreach name="item1">
    <where name="item2">
      <get name="SubGrid" />
      <compare op="&gt;">
        <cellget prop="Value" name="Allocation" type="double" failvalue="0"
parseformatting="true">
          <get name="item2" />
        </cellget>
        <const value="100.0" />
      </compare>
    </where>
    <cellset prop="Message" name="Allocation">
      <get name="item1" />
      <const value="Allocation must be less than 100%" />
    </cellset>
  </foreach>
</table>
```

Example #4 – Aggregate Validation

1. In some situations, we may need to ensure that the sum of all fields in a column add up to a specific amount. In the example below, we add up all allocations and ensure that they add up to 100%.

```
<table type="Grid" name="SubGrid" dataitemid="SubAccount_Grid">
  <if>
    <condition>
      <compare op="!=">
        <sum>
          <select name="allocationAmt">
            <get name="SubGrid"/>
            <cellget prop="Value" name="Allocation" type="double"
failvalue="0" parseformatting="true">
              <get name="allocationAmt" />
            </cellget>
          </select>
        </sum>
        <const value="100.0" />
      </compare>
    </condition>
    <postmessage dataitemid="SubAccount_Total">
      <const value="Total allocations must be 100%." type="String" />
    </postmessage>
  </if>
</table>
```

Rule Node Reference

Node Name: prop

Description: Represents a property or a cell in a table. The name is used to specify which one it represents.

Attributes:

1) name: The name of the cell or property to be set.
2) (optional) format: The formatting provided for the first parameter

Parameters:

1) [string] The value for the cell/prop
2) (optional) [bool] The isDisabled state for the property/cell

Node Name: table

Description: Opens and then saves the table stored in a dataitem for manipulation. When exiting the node it will save all changes to the dataitem and trigger its change event. Any messages set will be posted to the validation collection.

Attributes:

1) type: The type of table. Currently, only Grid is supported.
2) list: The name of the custom list to which a grid is bound.
3) dataitemId: The DataItem that holds the grid, i.e. the name of the field set in the designer.
4) name: The local variable containing the loaded list of rows that represent the table.

Parameters: any nodes.

Node Name: propset

Description: Sets a property on the loaded table. It only works within the context of the table node.

Attributes:

1) name: The name of the property of the table.
2) prop: The property on the item (ListName).

Parameters:

1) [table] The variable of the loaded table.
2) [any] The value to set the property.

Node Name: propget

Description: Gets a property on the loaded table. It only works within the context of the table node.

Attributes:

1) name: The name of the property of the table.
2) prop: The property on the item (ListName).

Parameters:

1) [table] The variable of the loaded table.

Node Name: cellget

Description: Gets the value of a cell in a row. This only works in the context of the table node. A row must be accessed through a list node on the loaded table, such as foreach or listget.

Attributes:

- name: The name of the column of the row.
- prop: The property on the item (Value, IsDisabled, Message).
- (optional) type: The type the value will be parsed into (int, bool, double, etc.).
- (optional) failvalue: The value to return if the parsing fails.
- (optional) parseformatting: Indicates if special formatting should be removed in the parse.

Parameters:

1) A row object pulled from a loaded table.

Node Name: cellset

Description: Sets the value on cell in a row. This only works in the context of the table node. A row must be accessed through a list node on the loaded table like foreach or listget.

Attributes:

- name: The name of the column of the row
- prop: The property on the item (Value, IsDisabled, Message)
- (optional) format: How to format the input. Only applies to Message and Value properties.

Parameters:

1) A row object pulled from a loaded table.
2) The value to set for the specified property of the cell.

Node Name: select

Description: Allows Selection and transformation of data in a list. It goes through the list and lets the second parameter change it. I.e. return a column in a row or modify a number.

Attributes:

- name: The local variable name where the current item in the list will be placed.

Parameters:

1) A list.
2) The transformation. The result of this will be stored in the list that is returned by the node. It can access the current item stored in name.

Node Name: where

Description: Reduces a list based on the condition. The first node is the list to be reduced. The second node must return a Boolean.

Attributes:

- name: The local variable name where the current item in the list will be placed.

Parameters:

1) A list
2) The node with returns if the current item is added. It can access the current item stored in name.

Areas Impacted

System Area	Yes	Comment
Admin Tool		
- Form Library		
- Design Forms	X	Wizard Designer
- Profile Administration		
- Reports		
- Deployment		
FireLight App		
- New Application		
- Edit Application	X	
- Signature Process		
- Review Queue		
- Manual Review		
- User Preferences		
- Inbound Integration		
- Outbound Integration		
- PDF Generation		
- Email System		
FireLight Console		
- Windows		
- iOS		
Other Systems		
- DTCC Integration		
- Commission Netting		
- Activity Reporting		