OPENXTALK DATA GRID GUIDE
Data Grid Guide v1.0, by LiveCode Ltd, edited and debranded for open source as requested by LiveCode Ltd.

==========

CHAPTER 1) INTRODUCTION
Data Grid, Table, Form, Create Table, Create Form, Customize Form, Tips.

Data grids display data in either the default Table or Form styles. Customize a data grid to include any other object. Data grids can provide a view into source data and display large data sets.

1.1) What Is A Data Grid?
A data grid integrates tables and lists (forms) in a stack. They combine nested groups and button behaviors to provide a flexible stylish way to display data. Data Grids are referred to as group or grp.

```
set the dgProp["column labels"] of grp "DataGrid 1" to "State" &cr& "Code"  --set data grid table column labels
put the dgText["true"] of group "DataGrid 1" into field "Text"  --data grid table text including headers
put the dgText of group "DataGrid 1" into field "Text"  --data grid table text without headers or data grid form text.
```

When the first data grid is dragged from the Tools palette to a stack, a substack is also created with two cards; card id 1002 and a second card containing a "row template" group for layout, and a "row behavior" script (if form) for controlling that data grid.

A new second data grid dragged from the Tools palette creates another new card on the same substack, the new card also containing a row template group, and row behavior script (if form) for the second data grid. There is a unique substack card for each data grid table and form in the main stack. Even though data grid tables don't require substack cards, one is created anyway in case controls like buttons or images are used or it's converted to a data grid form.

1.2) Data Grid Table
The default data grid table is an object similar to a table field that displays tabbed data in a stylish table complete with headers, sorting, and column alignment. A column can contain text by default, and graphics, images, objects with a customized column template and column behavior.

```
put the dgText["true"] of grp "DataGrid 1" into tData  --true/false, true includes headers in line 1
set the itemDel to tab  --tab delimited data
--make changes to tData here
set the dgText["true"] of grp "DataGrid 1" to tData
send "RefreshList" to grp "DataGrid 1"
```

1.3) Data Grid Form
The data grid form is a complex object similar to a list field that displays records the user can select. The data grid form uses a group as a template for each record. A record can contain text by default, and graphics, images, objects with a customized row template and row behavior. Data grid forms do not have headers. A data grid form is very fast because it only draws the records currently visible on screen, so large record sets can be used. Nearly any list can benefit from being displayed in a data grid form.

o A default data grid form requires non-tabbed line list data.
o A data grid form customized in its row template and row behavior will accept tabbed line list data.

1.4) How To Create A Data Grid Table
Data grid tables are used to display rows of data in structured columns.
1. Drag a data grid from the Tools palette to a stack.
2. Select it, Object > Object Inspector (or double-click it), and give it a name.
3. In the Columns pane, add column headers using the "+" button, renaming Col 1 to "State", and Col 2 to "Code".
4. In the Contents pane, paste or enter a tab delimited line list of text. Then ctrl-s to save.

```
Alabama	AL
Alaska	AK
Arizona	AZ
Arkansas	AR
```

The tab delimited text entered in the field will appear as columns in the data grid table. The Inspector automatically creates a data grid column for each text column entered. Clicking a triangle in the header row sorts that column. Column widths can be resized by dragging the column dividers in the header, and the entire data grid can then be resized in edit mode.

Populate Data Grid Using dgText Property
Default data grids can also be populated by code with the dgText property. For example:

```
on mouseUp
  put "State" &tab& "Code" &cr& \
  "Alabama" &tab& "AL" &cr& \
  "Alaska" &tab& "AK" &cr& \
  "Arizona" &tab& "AZ" &cr& \
  "Arkansas" &tab& "AR" into tText
  set the dgText["true"] of group "MyDataGrid" to tText    --true/false, true includes headers in line 1
end mouseUp
```

```
set the dgText["true"] of grp "DataGrid 1" to tData  --insert data in default data grids
put the dgText["true"] of grp "DataGrid 1" into tData  --extract data from any data grid
```

To insert data into a customized data grid, the dgData and an array must be used.

Populate Data Grid Using dgData Property

Data grids can also be populated by code using array data with the dgData property. This is more complex. For example:

```
on mouseUp
  put "Alabama" into tData[1]["State"]
  put "AL" into tData[1]["Code"]
  put "Alaska" into tData[2]["State"]
  put "AK" into tData[2]["Code"]
  put "Arizona" into tData[3]["State"]
  put "AZ" into tData[3]["Code"]
  put "Arkansas" into tData[4]["State"]
  put "AR" into tData[4]["Code"]
  set the dgData of group "MyDataGrid" to tData
end mouseUp
```

1.5) How To Create A Data Grid Form

Data grid forms are complex but less rigid than columns in a table. They offer control over the display of each record. Data grid forms don't have columns or column headers.

1. Drag a data grid from the Tools palette to a stack.
2. Select it, Object > Object Inspector (or double-click it), give it a name, and change the style to Form.
3. In the Contents pane, paste or enter a line list of text (no tabs for default form). Then ctrl-s to save.

```
Alabama-AL
Alaska-AK
Arizona-AZ
Arkansas-AR
```

To customize a data grid form, the "Row Template" layout and "Row Behavior" button script must be edited. Fortunately, some of the layout and much of the script is pre-written requiring some custom modification. This is the complex portion of a data grid form and not for beginning coders. Refer to the Data Grid Manual for detailed instructions.

Row Template Layout

Once programmed, the data grid form must be populated with data. A customized data grid form will accepts tab-delimited text items in return-delimited lines in the contents section of its properties Inspector, or populates by script with either that data or data converted to an array. Customize the row template layout with a check box button or additional field added to the "gray" group at the top left. Each object added corresponds to a data item.

Row Behavior Button Script

Once populated, the data grid form may respond to user interaction with a mouseUp handler in the row behavior script.

```
on mouseUp pMouseBtn  --user click
  local tData
  if pMouseBtn = "1" then  --left click
    put the dgDataOfLine[the dgHilitedLines of me] of me into tData  --array
    set the dgHilitedLines of me to empty  --remove selection
    answer tData["Label 1"] with "OK" titled "DataGrid Line Text"  --text of row clicked, default field "Label" in array tData["Label
1"]
  end if
  pass mouseUp
```

end mouseUp

1.6) Customize A Data Grid Form
Enable word wrap for a data grid form with four steps.
1. In the data grid form's Inspector, uncheck Fixed Control Height (off).
2. In "Row Template" layout, in msg box: set the dontWrap of fld "Label" to "false" --allow wrap
3. In "Row Behavior" script, in the LayoutControl handler, in the empty line above "set the rect of graphic" add two lines:
put (the top of fld "Label" of me + the formattedHeight of fld "Label" of me -5) into item 4 of theFieldRect --new bottom
set the rect of fld "Label" of me to theFieldRect --resize
4. Click Apply, save with ctrl/cmd-S. In msg box:
set the dgText of grp "DataGrid 1" to fld "MyLineList" --populate data grid form
---

1.7) Data Grid Tips
o Don't paste a data grid. Instead drag a new data grid from the Tools palette and paste in the template and script from the original. An alternative is pasting the card containing the data grid. Pasting a data grid on the same stack or a different stack doesn't create a new card in the data grid substack, the original's substack card is used instead.

o If the mainStack script has preOpenStack, openStack, preOpenCard, or openCard handlers, use "if the owner of this stack = empty then" to stop a data grid substack from triggering commands meant for the mainStack. An alternative is to place the "open" handlers in the first card's script.

o RefreshList and ResetList are data grid only commands used following changes. A data grid looks like a field but is a group.
send "RefreshList" to group "DataGrid 1" --following changes to data grid's data.
send "ResetList" to group "DataGrid 1" --following changes to data grid's row template group objects.

o The LayoutControl handler in the row behavior script is a common source of errors. It uses item 1,2,3,4 of the rect property to align the row template controls. That's Left,Top,Right,Bottom of the control rectangle.
set the left of fld "Label" of me... --re-position only
set the rect of fld "Label" of me... --resize

```
on LayoutControl pControlRect  --rect=L,T,R,B
  local theFieldRect
  put the rect of fld "Label" of me into theFieldRect
  put item 1 of pControlRect into item 1 of theFieldRect  --left
  put item 3 of pControlRect into item 3 of theFieldRect  --right
  set the rect of fld "Label" of me to theFieldRect
  set the rect of graphic "Background" of me to pControlRect  --resize
end LayoutControl
```

o Put a mouseUp handler in the row behavior script for a data grid to respond to a user click.

o The order of items in the dgText[ ] property is determined by the alpha order a-z of the data grid keys/columnHeaders, not by the row template or the row behavior script.

o Change the contents of a data grid with the dgText[ ] property. It has tab-delimited items and return-delimited lines. dgText["true"] includes the data grid keys/columnHeaders in line 1. The dgText and the dgText["false"] do not include keys/headers.

o Change the properties of a data grid with the dgProp["xx"] custom property. One useful property to change from default true to false is "scroll selections into view". In msg box:
set the dgProp["scroll selections into view"] of grp "DataGrid 1" to "false" --stop scroll jump

o When deleting a data grid, a dialog asks to leave or delete its data grid substack card "record template". Default "No" leaves

the orphan card in the data grid substack. "Yes" deletes the card, and the substack too if it's the only card. Choose "Yes" option.
==========

CHAPTER 2) FUNDAMENTALS

Stack "Data Grid Templates", Row Template, Column Template, Populate A Data Grid, Customize A Form, Customize A Table.

2.1) Stack "Data Grid Templates"

A data grid relies on templates to display data. These templates are controls used to visualize data. To manage these controls the IDE creates a stack named "Data Grid Templates" the first time a data grid is added to a stack. To see the stack, choose Tools > Project Browser, click "+".

This stack contains a single card for each data grid added to a stack. Each card contains the templates and behavior button(s) the data grid will use to display the data. The row template is the group that contains either the column template for a table or the controls for a form. The IDE assigns the row template group to the dgProps["row template"] property of the data grid.

The "Behavior Script" button is assigned as the behavior to the row template group and is used for data grid forms. This script controls how data is inserted into the row template controls and how those controls are positioned.

Opening The Data Grid Templates Stack

Select the data grid, choose Object > Object Inspector, click the "Row Template" button. The card containing the template for the selected data grid will open. Data grid templates can be customized to better represent data. A row template is a group that will be copied and used to draw data on the screen.

2.2) Row Template

The row template is a group control. This group represents a single record in the data being displayed. It can contain any control. When a data grid displays data it copies the row template into itself. It makes just enough copies to cover the visible area and then inserts data into these copies of the row template. As the user uses the scrollbar to scroll through the data the same copies are used but new data is inserted. This means the data grid is never drawing more records than the user can actually see making the data grid fast.

Data grid tables and forms use templates differently. A table uses a template to represent each column of data. A form uses a template to represent each row of data. A table template can also use a field, graphic, or button for a template.

2.3) Populate A Data Grid With Data

There are three ways to insert data into a data grid.

o Use The Property Inspector

In the Inspector's Contents pane, paste or type tab delimited line list of text into the field.

o Set The dgText Property

```
set the dgText["true"] of group "DataGrid 1" to tText  --tab delimited line list, line 1 = headers/keys
set the dgText["false"] of group "DataGrid 1" to tText  --tab delimited line list, no headers/keys
set the dgText of group "DataGrid 1" to tText  --default false, tab delimited line list, no headers/keys
```

For a table, set dgText["true"] and provide the header that maps each item of each line to a specific column. Customize a column with a column template layout and column behavior script to display imported data correctly.

Note: Named columns must already exist in a data grid table in order to be displayed, setting the dgText["true"] does not create the headers. Enter column names in the Inspector first. If table data contains more items than columns, new columns are added, named "Col 1", "Col 2"....

For a form, set dgText["false"] and each item of each line is named "Label 1", "Label 2", ... in the array that is passed to the FillInData handler. Customize the form with the row template layout and row behavior script to display imported data correctly.

o Set The dgData Property
The dgData property of a data grid is a multi-dimensional array and is the actual format that the data grid works with under the hood. The data grid expects the first dimension of the array to be integers. The number of keys in the first dimension represents the number of records being displayed in the data grid. Then store all data in the second dimension. Two examples:

```
on mouseUp  --dgData example 1
  local tData
  put "John" into tData[1]["FirstName"]  --tData is an array
  put "Smith" into tData[1]["LastName"]
  put "Mike" into tData[2]["FirstName"]
  put "Wilson" into tData[2]["LastName"]
  put "Susan" into tData[3]["FirstName"]
  put "Harris" into tData[3]["LastName"]
  set the dgData of group "DataGrid 1" to tData  --populate data grid
end mouseUp
```

```
on mouseUp  --dgData example 2
  local tMyList, tData, tCount = "1"
  put "John,Smith" &cr& "Mike,Wilson" &cr& "Susan,Harris" into tMyList
  repeat for each line i in tMyList
    put item 1 of i into tData[tCount]["FirstName"]  --tData is array
    put item 2 of i into tData[tCount]["LastName"]
    add 1 to tCount
  end repeat
  set the dgData of group "DataGrid 1" to tData  --populate data grid
end mouseUp
```

Important: if using a data grid table, a key in the array must match the header of a column in the table. A column header "FirstName" must have a corresponding key named "FirstName"
tData[1]["FirstName"]

2.4) Customize A Form's Row Template And Row Behavior
Adding a data grid form to a stack automatically creates a row template layout and a row behavior script. This template by default will display a line list of text. The row template and row behavior can be customized with additional controls and code to accept tabbed line list data.

Step 1: In a data grid form's Inspector, click the "Row Template" button. Ensure Edit > Select Grouped Controls is unchecked (off). Select the gray row group in the upper left, Object > Edit Group. The row template can now be customized with additional controls like a field or image area.

Step 2: Any controls added must have row behavior code associated with them. Click the "Row Behavior" button to add code to the handlers associated with this form to display data for a new control. The "Row Behavior" button contains several message handlers that fill and align controls in the row template. The most important handlers are FillInData, ResetData, and LayoutControl. Two examples:

```
on FillInData pDataArray  --example 1
  set the text of field "Name" of me to pDataArray["LastName"] &", "& pDataArray["FirstName"]
end FillInData
```

Note: the pDataArray parameter passed to FillInData for a form is an array representing the current row being displayed. This is different than the pData parameter passed to FillInData for a table column template which is the value of the column.

Important: Always refer to controls in the template using "of me". Multiple copies of the template will be made with multiple controls all having the same name. Using "of me" removes any ambiguity and ensures the data is displayed in the correct control.

```
on LayoutControl pControlRect  --example 2
  local theFieldRect
  --resize graphic "Background" and fields, position objects with rectangle property: left,top,right,bottom = 1,2,3,4
  put the rect of field "Label" of me into theFieldRect
  put item 3 of pControlRect - 5 into item 3 of theFieldRect  --new right, border = 5 pixels
  set the rect of field "Label" of me to theFieldRect
  put (the top of fld "Label" of me + the formattedHeight of fld "Label" of me -5) into item 4 of theFieldRect  --new bottom
  set the rect of fld "Label" of me to theFieldRect  --resize field for word wrap
  --
  set the rect of graphic "Background" of me to pControlRect  --resize graphic
end LayoutControl
```

A Complete Customized Row Behavior

The LayoutControl handler is the most confusing. The goal is to resize controls for the data, align them as desired, and then resize the graphic "Background" to match the combined size of the controls. The pControlRect parameter is the rectangle of graphic "Background".

```
on LayoutControl pControlRect  --two aligned fields
   local theFieldRect
   --resize objects, position objects, and resize graphic "Background" with rectangle property: left,top,right,bottom = 1,2,3,4
   put the rect of field "Label" of me into theFieldRect  --field 1
   put item 3 of pControlRect - 5 into item 3 of theFieldRect  --new right
   set the rect of field "Label" of me to theFieldRect  --resize field
   --
   put the rect of field "Label2" of me into theFieldRect  --field 2
   put item 3 of pControlRect - 5 into item 3 of theFieldRect  --new right
   set the rect of field "Label2" of me to theFieldRect  --resize field
   --
   set the left of field "Label2" of me to the left of field "Label" of me  --align left
   set the top of field "Label2" of me to the bottom of field "Label" of me  --align 1 above 2
   --
   set the rect of graphic "Background" of me to pControlRect  --resize graphic
end LayoutControl

on FillInData pDataArray
  set the text of field "Label" of me to pDataArray["label 1"]
  set the text of field "Label2" of me to pDataArray["label 2"]
end FillInData

on ResetData
  set the text of field "Label" of me to empty
  set the text of field "Label2" of me to empty
end ResetData
```

Add field "Label2" in the row template (or copy/paste/rename field "Label"). Add some tabbed line list data to the Contents pane and uncheck Fixed Control Height (off).

```
John    Smith
Sally   Anders
Frank   Wilson
Sue     Harris
```

2.5) Customize A Table's Columns

Customizing a data grid table is actually more complicated than customizing a data grid form. A form has one row template and

one row behavior. A table has a row template (but not a row behavior), and can have a column template and column behaviors for any customized column.

Step 1: In the table Inspector's Column pane, select a table column to customize. Click the "+" button at the bottom of the pane to create a column template. (Not the "+" button at the top of the columns list.) The "Column Behavior" button next to it is enabled, and a column template is created. Don't click the "Column Behavior" button yet.

Step 2: Click in the new column template to bring it to the front. Ensure Edit > Select Grouped Controls is unchecked (off). Select the gray column group in the upper left, Object > Edit Group. The column template can now be customized with additional controls like an image, a graphic, or a button.

Step 3: Any controls added must have column behavior code associated with them. Click the enabled "Column Behavior" button to add code to the handlers associated with this table column to display data for a new control. The "Column Behavior" button contains several message handlers that fill and align controls in the column template. The most important handlers are FillInData, ResetData, and LayoutControl. An example:

```
on FillInData pData  --example 1, pData is not array data
    set the visible of btn "Star1" of me to pData >= "10"
    set the visible of btn "Star2" of me to pData >= "20"
    set the visible of btn "Star3" of me to pData >= "30"
end FillInData
====================
```

CHAPTER 3) HELP WORKING WITH DATA GRIDS

Get Selected Line, Get Data From Row/Column, Add Data, Update Data, Clear Data, Add MouseDown, Store Option Menu Value, Reset Template After Changes, Use Template In Multiple Data Grids, Internal Array, Sum Column Values, Focus, Export Data, Check Boxes, Find Line, Search.

3.1) Get The Selected Line

Get the selected line of a data grid with the dgHilitedLines or the dgLine of me properties.

```
put the dgHilitedLines of group "DataGrid 1"  --hilited line(s)

on mouseDown  --in row/column behavior script
  put the dgLine of me
end mouseDown
----
```

3.2) Get Data Associated With A Row Or Column

Get Data From A Row

Get the data array from a selected row with the dgDataOfLine or dgDataOfIndex properties.

o dgDataOfLine: order of rows as they appear.

o dgDataOfIndex: order of rows based on an internal index number unaffected by sorting.

```
on mouseUp pBtnNum  --in row behavior script
  local tLine, tData
  if pBtnNum is 1 then
    put the dgHilitedLines of group "DataGrid 1" into tLine
    put the dgDataOfLine[tLine] of group "DataGrid 1" into tData  --array
    answer tData["FirstName"] && tData["LastName"] with "OK" titled "You Clicked..."  --selected row data
  end if
end mouseUp
----
```

Get Data Of A Column From A Row

Get the data of a particular column cell from a row with the getDataOfLine() or getDataOfIndex() functions.

o getDataOfLine(): returns cell value of a key/column for the selected row.

o getDataOfIndex(): returns cell value of a key/column for the selected row.

```
getProp cSelectedName  --in data grid's group script, create a custom property
  local tLine
  put the dgHilitedLines of me into tLine
  return getDataOfLine(tLine, "LastName")  --"LastName" is an array key
end cSelectedName
```

Access this custom property from any script in the data grid's stack:

```
put the cSelectedName of group "DataGrid 1" into tUserSelection
```

Get Data When The Selection Changes

Get the data of a new selection with the selectionChanged message.

```
on selectionChanged pHilitedIndex, pPrevHilitedIndex  --in data grid's group script
  local tData
  put the dgDataOfIndex[ pHilitedIndex ] of me into tData  --an array
  ViewRecordOfName tData["LastName"]  --custom command in card script
end selectionChanged
```

Get Data In A Row Behavior

Get the data in a row behavior script with the dgLine or dgIndex properties. The dgIndex is the numeric level 1 index used when setting the dgData property.

```
on mouseUp pMouseBtnNum  --example 1, in data grid form's row behavior script
  local tData
  if pMouseBtnNum is "1" then  --left-click
    put the dgDataOfIndex[ the dgIndex of me] of the dgControl of me into tData  --an array
    ViewRecordOfName tData["LastName"]  --custom command in card script
  end if
end mouseUp
```

To move this mouseUp code into the data grid's group script, change the me references to target and ensure the user clicked on a row. This example uses getDataOfIndex() rather than dgDataOfIndex to show an alternative.

```
on mouseUp pMouseBtnNum  --example 2, in data grid form's group script
  local tData, tIndexClicked, tName
  if pMouseBtnNum is "1" then  --left-click
    put the dgIndex of the target into tIndexClicked
    if tIndexClicked is not empty then
      put GetDataOfIndex(tIndexClicked, "LastName") into tName
      ViewRecordOfName tName
    end if
  end if
end mouseUp
```

Get Data In A Column Behavior

Get the data in a column behavior script with the dgLine or dgIndex, and the dgColumn properties.

```
on mouseUp pMouseBtnNum  --example 1, in data grid table column behavior script
  local tColumnValue
  if pMouseBtnNum is "1" then  --left-click
    put getDataOfIndex(the dgIndex of me, the dgColumn of me) into tColumnValue
    --do something with column value
  end if
```

```
end mouseUp
```

To move this mouseUp code into the data grid's group script, change the me references to target and ensure the user clicked on a column.

```
on mouseUp pMouseBtnNum  --example 2, in data grid table's group script
  local tColumn, tColumnValue
  if pMouseBtnNum is "1" then  --left-click
    put the dgColumn of the target into tColumn
    if tColumn is not empty then  --user clicked a column
      put getDataOfIndex(the dgIndex of the target, tColumn) into tColumnValue
      --do something with column value
    end if
  end if
end mouseUp
```

3.3) Add A Row Of Data To An Existing Data Grid

Add a row of data to a data grid by setting the dgText or dgData property, by using the AddLine or AddData commands, or directly by entering it in the Contents pane of the data grid.

Using AddLine

Use AddLine to create a tab delimited string of text containing the values for a new row and have it appear at a specific line.

```
on mouseUp  --in button script
  local tData, tColumns, tRow
  put "Alice" &tab& "Bennett" &tab& "Manager" into tData  --tab delimited data
  put "FirstName" &cr& "LastName" &cr& "Title" into tColumns  --return delimited column headers
  put the dgNumberOfLines of me + 1 into tRow  --insert data after last row
  dispatch "AddLine" to group "DataGrid 1" with tData, tColumns, tRow
  ScrollLineIntoView tRow  --scroll new row into view
end mouseUp
```

Using AddData

Use AddData to create an array containing the values for a new row, and have it appear at a specific line.

```
on mouseUp  --in button script
  local tData, tRow
  put "Alice" into tData["FirstName"]  --an array
  put "Bennett" into tData["LastName"]
  put "Manager" into tData["Title"]
  put "3" into tRow  --insert data at row 3
  dispatch "AddData" to group "DataGrid 1" with tData, tRow
  put the result into tResult  --integer if successful, error string otherwise
  if tResult is an integer then ScrollIndexIntoView tResult  --scroll new row into view
end mouseUp
```

Scroll Data Into View

To scroll the new row of data into view, use the ScrollLineIntoView or ScrollIndexIntoView commands.

```
ScrollLineIntoView tRow
if tResult is an integer then ScrollIndexIntoView tResult
```

3.4) Update Data In A Row

To update a row of data and refresh automatically, use the dgDataOfLine or dgDataOfIndex properties. Set either property to an

array containing the values for the row.

```
on mouseUp  --in button script, new data for hilited line
  local tData
  put "New First Name" into tData["FirstName"]
  put "New Last Name" into tData["LastName"]
  put "New Title" into tData["Title"]
  set the dgDataOfIndex[ the dgHilitedIndex of group "DataGrid 1" ] of group "DataGrid 1" to tData
end mouseUp
```

To update a row of data without refresh, use SetDataOfIndex command, then RefreshIndex command to redraw the data grid.

```
on mouseUp  --in button script, new data for hilited line
  local tData
  put "New First Name" into tData["FirstName"]
  put "New Last Name" into tData["LastName"]
  put "New Title" into tData["Title"]
  dispatch "SetDataOfIndex" to group "DataGrid 1" with the dgHilitedIndex of group "DataGrid 1", empty, tData
  dispatch "RefreshIndex" to group "DataGrid 1" with the dgHilitedIndex of group "DataGrid 1"
end mouseUp
```

Example that sets individual keys of the row one at a time:

```
dispatch "SetDataOfIndex" to group "DataGrid 1" with the dgHilitedIndex of group "DataGrid 1", "FirstName", "New First Name"
dispatch "SetDataOfIndex" to group "DataGrid 1" with the dgHilitedIndex of group "DataGrid 1", "LastName", "New Last Name"
dispatch "SetDataOfIndex" to group "DataGrid 1" with the dgHilitedIndex of group "DataGrid 1", "Title", "New Title"
dispatch "RefreshIndex" to group "DataGrid 1" with the dgHilitedIndex of group "DataGrid 1"
```

3.5) Clear Data From A Data Grid

To clear all data from a data grid, set the dgText or the dgData of the data grid to empty.

```
set the dgText of group "DataGrid 1" to empty  --clear all data
```

3.6) Add A MouseDown Event To A Data Grid

Use a mouseDown handler in a data grid's group script to show a contextual menu or to use data in the row the user clicked. Adding a mouseDown handler intercepts the message the data grid normally handles. This changes the behavior of the data grid. The data grid behavior script processes mouseDown AFTER the custom mouseDown handler you placed in the data grid's group script.

---

dgMouseDown

To work around this the data grid wraps all its mouseDown functionality in a dgMouseDown handler. The line the user clicks on will be selected before your custom mouseDown code executes. The following examples show how to code a mouseDown handler in a data grid's group script.

```
on mouseDown pMouseBtnNum  --example 1, in data grid's group script, for contextual menu
  dgMouseDown pMouseBtnNum  --allow data grid to process mouseDown first and select row clicked
  if pMouseBtnNum is "3" then  --right-click for contextual menu
    popup button "MyContextualMenu"
  end if
  --don't pass mouseDown
end mouseDown
```

```
on mouseDown pMouseBtnNum  --example 2, in data grid's group script, value from data grid table cell clicked
  local tColumnValue
  dgMouseDown pMouseBtnNum  --allow data grid to process mouseDown first and select row clicked
```

```
  put getDataOfIndex(the dgHilitedIndexes of me, the dgColumn of the target) into tColumnValue
  --don't pass mouseDown
end mouseDown
```

Note: the mouseDown message is not passed after calling dgMouseDown. This would only repeat the call to dgMouseDown.

3.7) Store An Option Menu Value When the User Makes A Selection
In a data grid table, in column "Rating", create a column template and column behavior to be customized for an option menu button. The goal is to update the data associated with a row to reflect the selection the user makes in the option menu button.

Edit Column Template
Step 1. Drag option menu button "Rating" to a card, replace Choice 1, Choice 2... with Best, Good, and Poor on separate lines. Select/copy the button.
Step 2. In the table Inspector's Columns pane, select column "Rating" and click the "+" button at the bottom of the pane.
Step 3. In the column template, ensure Edit > Select Grouped Controls is unchecked (off). Select the "gray" group in the upper left, Object > Edit Group.
Step 4. Now click the "Rating" group, Edit > Edit Group. Paste the button into the "Rating" group. Position the button within the "gray" graphic to the right of "Rating". Object > Stop Editing Group.

Edit Column Behavior
Step 1. In the table Inspector's Columns pane, select column "Rating" and click the "Column Behavior" button at the bottom of the pane.
Step 2. Replace the existing FillInData and LayoutControl handlers, and add a menuPick handler with the following code:

```
on FillInData pData
  lock messages  --block menuPick when setting menuHistory
  set the menuHistory of btn 1 of me to lineOffset(pData, the text of btn 1 of me)  --user choice
  unlock messages
end FillInData

on LayoutControl pControlRect
   set the rect of btn 1 of me to pControlRect  --resize button
end LayoutControl

on menuPick pChosenItem
  SetDataOfIndex the dgIndex of me, the dgColumn of me, pChosenItem
end menuPick
```

The Behavior In Action
To verify the data grid table updates, change a few of the buttons' choices, and look in the table Inspector's Contents pane.

3.8) Reset A Data Grid After Template Changes
Reset a data grid after row template changes with the ResetList command.
In msg box: dispatch "ResetList" to group "DataGrid 1" --reset data grid after template changes

```
on mouseUp  --in button script, update text of a button in data grid table
  local tRowTemplate, tColTemplate
  put the dgProps["row template"] of group "DataGrid 1" into tRowTemplate  --reference to row template
  put the long id of group "Rating" of tRowTemplate into tColTemplate  --reference to table column template
  set the text of button 1 of tColTemplate to "Best" &cr& "Better" &cr& "Good"  --update text
  dispatch "ResetList" to group "DataGrid 1"  --reset data grid
end mouseUp
```

3.9) Use A Template In Multiple Data Grids
If a stack needs to use the same style of data grid on multiple cards, they can share the same customized row template.

Step 1. Create the first data grid and customize the row template. The row template used is identified with the dgProps["row template"] property. Use the message box to get a reference. In message box:
put the dgProps["row template"] of group "DataGrid 1" --returns long id of row template

Step 2. Add another data grid to the stack, and in message box set the dgProps["row template"] property to the long id of the first data grid's row template. In message box:
set the dgProps["row template"] of group "DataGrid 2" to group id 1003 of card id 1002 of stack "Data Grid Templates 1237599031838"

The row template can be shared with a data grid in another open stack. In a button or message box:

```
on mouseUp
  set the dgProps["row template"] of group "DataGrid 1" of stack "MyStack 1" \
  to the dgProps["row template"] of group "DataGrid 1" of stack "MyStack 2"  --share row template in different open stack
end mouseUp
```

3.10) Quick Check Of Data Grid's Internal Array
For a quick check of a data grid's internal array, use the PrintKeys command. In the message box send "PrintKeys" to a data grid to return the first line of each key in the internal data array as text. In message box:
send "PrintKeys" to group "DataGrid 1" --returns data array as text

For direct access to the data, check in the data grid Inspector's Contents pane.

3.11) Sum Column Values
Use a custom property in a data grid table or form to return the sum total of values in a column.

Step 1. Place the following code in a data grid's group script, usually placed in a table but ok in a form.

```
getProp cSumColumn [pColumn]  --in data grid's group script
  local tTotal = "0", tData
  put the dgText of me into tData
  set the itemDel to tab
  repeat for each line i in tData
    if item pColumn of i is a number then add item pColumn of i to tTotal  --pColumn = column number
  end repeat
  return tTotal  --sum total in column
end cSumColumn
```

Step2. Use the custom property for a data grid with numbers in a column. Specify the parameter column (item) number quoted in brackets. In this example, numbers are in column 1. In a button to put sum into a field or dialog, or in message box:
put the cSumColumn ["1"] of group "DataGrid 1" --custom property, specify column number to sum
---

3.12) Determine If A Data Grid Has Focus
To determine if a data grid is the object receiving user keystrokes:

```
if the long id of group "DataGrid 1" is in the long id of the focusedObject then  --focus
  --do something here
end if
```

3.13) Export Data From A Data Grid
To get data out of a data grid, use the dgText or dgData properties.

```
on mouseUp  --example 1, export using dgText (text)
  get the dgText of group "DataGrid 1"  --tabbed line list
  replace tab with ", " in it  --tab to comma delimited
  put it into message box
end mouseUp

on mouseUp  --example 2, export using dgData (array)
  local tData, tIndexes, tText
  put the dgData of group "DataGrid 1" into tData
  put the dgIndexes of group "DataGrid 1" into tIndexes  --comma delimited list of dgData keys
  repeat for each item i in tIndexes
    put "#:" && tData[i]["#"] &cr after tText
    put "State:" && tData[i]["State"] &cr after tText
    put "Code:" && tData[i]["Code"] &cr&cr after tText
  end repeat
  delete char -2 to -1 of tText  --last two returns
  put tText into message box
end mouseUp
```

3.14) Check Box In A Data Grid
To add a check box to a data grid form, the row template layout and row behavior script must be customized.

Row Template Layout
Step 1. In a form Inspector, click the "Row Template" button. Ensure Edit > Select Grouped Controls is unchecked (off). Select the "gray" group in the upper left, Object > Edit Group.
Step 2. Drag a check box from the Tools palette to the left edge of the "gray" group. Object > Stop Editing Group.
Step 3. To edit the check box, Edit > Select Grouped Controls. Select the check box and in its Inspector, name "Check", uncheck showName, width = 23, height = 21, left/top = 0, and make any final positioning of the existing "Label" field.

Row Behavior Script
Step 4. In the form Inspector, click the "Row Behavior" button. Replace the existing FillInData, LayoutControl, and ResetData handlers with these new ones, and add the new mouseUp handler. Then save changes with ctrl/cmd-S.

```
on FillInData pDataArray  --array keys
  set the hilite of btn "Check" of me to pDataArray["Check"]
  set the text of fld "Label" of me to pDataArray["Label 1"]
end FillInData

on LayoutControl pControlRect  --row template layout, rect=L,T,R,B
  local theFieldRect
  set the left of btn "Check" of me to (item 1 of pControlRect)  --left
  put the rect of fld "Label" of me into theFieldRect
  put the right of btn "Check" of me -5 into item 1 of theFieldRect  --left
  put item 3 of pControlRect into item 3 of theFieldRect  --new right
  set the rect of fld "Label" of me to theFieldRect  --resize
  set the rect of graphic "Background" of me to pControlRect
end LayoutControl

on ResetData  --control no longer being used to display data
  set the hilite of btn "Check" of me to "false"
  set the text of fld "Label" of me to empty
```

```
end ResetData

on mouseUp pMouseBtnNum  --user click
  if pMouseBtnNum = "1" then  --left click
    if the short name of target() = "Check" then SetDataOfLine the dgLine of me,"Check",the hilite of target()  --check box t/f
  end if
end mouseUp
```

Add Data To Data Grid Form
Step 5. Populate the data grid form with a button.

```
on mouseUp  --populate customized data grid form
 local tText
 put "Check" &tab& "Label 1" &cr& \
 "false" &tab& "Pay Bills" &cr& \
 "true" &tab& "Wash Clothes" &cr& \
 "false" &tab& "Pickup Milk" into tText  --tabbed line list
 set the dgText["true"] of grp "DataGrid 1" to tText  --["true"] = keys in first line
end mouseUp
```

Uncheck All Check Boxes In A Data Grid
To uncheck all the check boxes, use the dgNumberOfLines property and the SetDataOfLine command. In a button script:

```
on mouseUp  --in button script, uncheck all check boxes
 repeat with i = 1 to the dgNumberOfLines of group "DataGrid 1"
   dispatch "SetDataOfLine" to group "DataGrid 1" with i, "check", "false"
 end repeat
 dispatch "RefreshList" to group "DataGrid 1"
end mouseUp
```

Count Check Box Selections In A Data Grid
To count the check box selections, use the filter command. Display the result or a subset of the result. In a button script:

```
on mouseUp  --count check box selections
  local tData, tNewData
  put the dgText of grp "DataGrid 1" into tData  --no keys in line 1
  filter tData with ("*"& "true" &"*")  --keep checked lines, "*" allows match to any chars before or after string
  set the itemDel to tab
  put tData into tNewData
  replace tab with ", " in tNewData  --no subset
  --repeat for each line i in tData  --subset of line
    --put item 4 of i &", "& char 1 to 50 of item 3 of i &cr after tNewData  --subset of line
  --end repeat
  --delete char -1 of tNewData  --return
  answer "You selected" && number(lines in tNewData) && "check boxes:" &cr&cr& tNewData with "OK" titled "Check Box
Selections"
end mouseUp
```

3.15) Find Line The Mouse Is Over
To identify the data grid line the mouse is hovering over, use the dgDataControl and the dgLine or dgIndex properties. In msg box:
```
put the dgLine of the dgDataControl of the mouseControl  --position mouse, then returnKey

put the dgDataControl of the mouseControl into tControl
```

if tControl is not empty then put the dgLine of tControl into tLineNum
---

3.16) Search A Data Grid
To search for a word or string in a data grid, use the dgText property and the filter command. Display the result. In a button script:

```
on mouseUp  --search data grid
  local tSearchString, tData
  ask "Search for what word or string of words in the data grid?" with "lorem" titled "Search"
  if it = empty then exit to top
  put it into tSearchString
  put the dgText of grp "DataGrid 1" into tData  --array to text, no keys in line 1
  filter tData with ("*"& tSearchString &"*")  --keep lines containing tSearchString
  replace tab with ", " in tData
  if tData = empty then answer "Sorry, no matching data found." with "OK" titled "Search"
  else answer "Found" && number(lines in tData) && "lines that matched" &&quote& tSearchString &quote&"." &cr&cr& tData
with "OK" titled "Search"
end mouseUp
```
==========

CHAPTER 4) HELP WORKING WITH TABLES
Change Column Alignment, Column Sort, Resize Columns, Override Default Behavior, User Click In Table Header, Display Contextual Menu, HtmlText & RtfText & UnicodeText, Line Numbers, Customize Column Sort, Disable Column Sort, Action After User Sorts, Align Column Decimals, Color A Line, Display Cell Value.

4.1) Change A Column Alignment
To change the column alignment of a data grid table, in the Inspector's Columns pane, select the column to modify and change alignment to left, center, or right. To change the column alignment by code, in message box:

```
set the dgColumnAlignment["Amount"] of group "DataGrid 1" to "right"  --right justify column
set the dgHeaderAlignment["Amount"] of group "DataGrid 1" to "right"  --right justify header too
```

4.2) Change A Column Sort
To change the column sort of a data grid table, in the Inspector's Columns pane, select the column to modify and change the sort direction to ascending or descending. Change the sort type to text, numeric, or dateTime. To change the sort by code, in message box:

```
set the dgColumnSortDirection["Amount"] of group "DataGrid 1" to "descending"  --sort low-hi
set the dgColumnSortType["Amount"] of group "DataGrid 1" to "numeric"  --sort numeric
```

4.3) Resize A Column Width
To change the column width of a data grid table, in the Inspector's Columns pane, select the column to modify and change the width entry. To change the width by code, in message box:

```
set the dgColumnWidth["Amount"] of group "DataGrid 1" to "75"  --integer pixel width
```

4.4) Override Default Behavior Rendering Data To A Cell
By default, a data grid table uses a single field object for each cell in a table and assigns the text property of that field to the cell's data. Customize the default behavior with a button script. This can be used to render htmlText and unicode text, trailing off text too wide for a column, or coloring specific cells.

Create a button that will become a customized column behavior script. The button script will be used to fill in each cell in the table.
Step 1. Drag button "MyColBehavior" to a card with a data grid table. In message box:

set the script of button "MyColBehavior" to the script of button "Default Column" of stack "revDataGridLibrary" --default code

Step 2. Assign the table's column behavior to the button. In message box:
set the dgProps["default column behavior"] of group "DataGrid 1" to the long id of button "MyColBehavior" --behavior script

Step 3. Customize the button script.

Truncate Tail --example 1
To truncate the tail end of every cell whose content is too wide to fit, use the built-in custom command TruncateTail. The command parameters are the short id of a field and a string that signifies the text is being truncated. Replace the FillInData and LayoutControl handlers with the following code in button "MyColBehavior". Then in message box: send "RefreshList" to group "DataGrid 1"

```
on FillInData pData
   set the cText of me to pData  --custom property
   set the text of the long ID of me to the cText of me
   TruncateTail the short id of me, "..."  --custom command
end FillInData

on LayoutControl pControlRect
  set the text of the long ID of me to the cText of me
  TruncateTail the short id of me, "..."  --custom command
end LayoutControl
```

Note: TruncateTail can cause visual lag if there are lots of cells being displayed. To truncate specific columns, use the following code:

```
switch the dgColumn of me
 case "Col 1"  --truncate only
 case "Col 2"  --truncate only
   TruncateTail the short id of me, "..."
 break
end switch
```

Color Cells --example 2
To dim any cell that is empty, use the field's opaque, backColor, and blendLevel properties. Add the following code in the FillInData handler, below the TruncateTail command. Then in message box: send "RefreshList" to group "DataGrid 1"

```
if pData = empty then
 set the opaque of me to "true"  --enable color
 set the backColor of me to "black"
 set the blendLevel of me to "50"  --50%
else
 set the opaque of me to "false"
 set the backColor of me to empty
 set the blendLevel of me to "0"
end if
```

4.5) Determine If User Clicks A Table Header
to determine if A user clicks in a table header, use the dgHeader and dgHeaderControl properties of the target control clicked.

The Table Header
o the dgHeader of the target: returns the long id of the group containing all header controls.
o the dgHeaderControl of the target: returns the long id of the group containing clicked column header controls.

To demonstrate these properties, temporarily place the following code in a table's group script and click at different header locations:

```
on mouseDown pBtnNum  --in table's group script
  put "dgHeader:" && the dgHeader of the target &cr&cr& "dgHeaderControl:" && the dgHeaderControl of the target
end mouseDown
```

4.6) Display Contextual Menu When User Clicks A Column Header

A contextual menu is displayed during a mouseDown message when the user right-clicks the mouse. To determine the click was on a table column header, use the dgHeaderControl of the target. Use the name of a table column appended with "Popup" to get the popup button's name.

Step 1: Drag a popup button from the Tools palette to near the data grid table, and name for a table column header && "Popup". Place the following code in the button's script:

```
on menuPick pItemName  --in pop-up button's script
  switch pItemName
    case "Choice 1"; put "You chose the first item."; break
    case "Choice 2"; put "That is the second choice."; break
    case "Choice 3"; put "You made..." && pItemName; break
  end switch
end menuPick
```

Step 2: Place the following script in the table's group script.

```
on mouseDown pBtnNum  --in table's group script
  local tBtnName
  dgMouseDown pBtnNum  --let data grid process mouseDown first
  if pBtnNum = "3" then  --right-click
    if the dgHeaderControl of the target is not empty then  --column header clicked
      put (the dgColumn of the target) && "Popup" into tBtnName  --build contextual btn's name
      if there is a button tBtnName then popup button tBtnName
    end if
  end if
end mouseDown
```

4.7) Working With htmlText, rtfText, Or unicodeText

By default all data grid table columns have their text property assigned. To work with htmlText, rtfText, or unicodeText a custom column behavior must be provided.

Option 1: Set the dgProp["Default Column Behavior"] Property

One way to change the default behavior of table columns is to use the dgProp["default column behavior"] property. Set this property to a button id whose script contains the custom column behavior. This overrides the default behavior provided by a button on the revDataGridLibrary stack.

Create a button that will become a customized column behavior script. The button script will be used to fill in each cell in the table.

Step 1. Drag button "MyColBehavior" to a card with a data grid table. In message box:

```
set the script of button "MyColBehavior" to the script of button "Default Column" of stack "revDataGridLibrary"  --default code
```

Step 2. Assign the table's column behavior to the button. In message box:

```
set the dgProps["default column behavior"] of group "DataGrid 1" to the long id of button "MyColBehavior"  --behavior script
```

Step 3. Customize the button script. Here is an example of a FillInData handler that takes a column value encoded as UTF-8 and sets the unicodeText of the column field.

```
on FillInData pData
  set the unicodeText of me to uniEncode(pData, "utf8")  --convert to unicode
end FillInData
```

Important: If you decide to write your own default behavior script make sure to include the dgDataControl getProp handler. This is required in order for the data grid to work properly.

```
getProp dgDataControl  --required by library to locate the control
  return the long ID of me
end dgDataControl
```

Note: To see the default column script the data grid table uses, in message box:

```
edit the script of button "Default Column" of stack "revDataGridLibrary"
```

Option 2: Create a Custom Column Template

Alternatively you can create your own custom column template for a particular column. This allows complete control over the look and feel of the column.

4.8) Display Line Numbers In A Table

Step 1. In a table Inspector's Columns pane, add a column with the "+" button and rename the column "Line Number".

Step 2. Use the UpArrow to move column "Line Number" to the top of the list, making it the first column of the table.

Step 3. Create a column template by clicking the "+" button at the bottom of the Inspector. Close and save the template.

Step 4. In the Columns pane, click the enabled "Column Behavior" button. Replace the existing FillInData and LayoutControl handlers with these:

```
on FillInData pData
  set the text of field 1 of me to the dgLine of me  --display line number
end FillInData

on LayoutControl pControlRect
  set the rect of field 1 of me to pControlRect
end LayoutControl
```

Step 5. In the table Inspector's Columns pane, click button "Refresh Data Grid". The line numbers appear in the table.

To set the dgText of a data grid table with first column line numbers, pass dgText["true"] and prepend the column names:

```
put "State" &tab& "Code" into tColNames
set the dgText["true"] of group "DataGrid 1" to tColNames &cr& tData  --line 1 contains line numbers
```

4.9) Custom Column Sort

To custom sort a data grid table column, use the SortDataGridColumn message sent when a user clicks a column. A table calls the built-in handler SortDataGridColumn whenever the dgProps["sort by column"] property is set. This includes when the user clicks on a column header to sort that column. By placing a modified version of the SortDataGridColumn in the data grid's group script, this message can be captured and sorting customized.

An example of a SortDataGridColumn handler:

```
on SortDataGridColumn pColumn  --in table's group script
  switch pColumn
    case "Line Number"
      MySort pColumn  --custom command
```

```
    break
   case "Zip Code"
    break  --prevent sort
   default  --all other columns
    pass SortDataGridColumn  --allow normal sort
    break
  end switch
end SortDataGridColumn

command MySort pColumn  --custom sort data
  local tData, tNewData, tDirection, tIndexes
  put the dgData of me into tData
  repeat for each key i in tData
   put i &tab& tData[i][pColumn] &cr after tNewData
  end repeat
  delete the last char of tNewData
  --
  set the itemDelimiter to tab
  put the dgColumnSortDirection[pColumn] of me into tDirection  --direction for this column
  if tDirection is "ascending" then sort lines of tNewData ascending by item 2 to -1 of each
  else sort lines of tNewData descending by item 2 to -1 of each  --sort all data by item 2
  --
  repeat for each line i in tNewData  --rebuild indexes
   put item 1 of i &comma after tIndexes
  end repeat
  delete the last char of tIndexes
  --
  set the dgIndexes of me to tIndexes  --new order
end MySort
```

4.10) Disable All Column Sorting
Step 1. To disable all sorting of all table headers, in message box:
```
set the dgProp["sort by column"] of group "DataGrid 1" to empty
```

Step 2. Add a SortDataGridColumn handler to the data grid table's group script to trap this message, not passing it will effectively disable sorting. Now a click in the table header will not sort the data.
```
on SortDataGridColumn pColumn
end SortDataGridColumn
```

Note: Once you add the SortDataGridColumn handler to a data grid table, setting the dgProp["sort by column"] property will no longer do anything. To later set the "sort by column" property, remove the SortDataGridColumn handler, or at least pass the message.

4.11) Perform Action After User Sorts
To perform an action after a data grid table is sorted, use the SortDataGridColumn handler. Call the handler but not pass it, so you can take additional actions. Place the following code in a table's group script.

```
on SortDataGridColumn pColumn
  SortByColumn pColumn  --allow table to sort normally
  --do additional actions here
  --don't pass SortDataGridColumn
end SortDataGridColumn
```

4.12) Align Decimals In A Column?
To right align decimal data for a column, use the numberFormat on number data before populating the data grid table. Use the dgText property to populate, and the dgHeaderAlignment and dgColumnAlignment properties to right align the numbers column. For a new data grid table, the "Amount" and "Item" headers must first be created in the Inspector's Columns pane.

```
on mouseUp  --in button
  local tData, tNewData, tHeaders
  put fld "NumberData" into tData  --tab delimited line list, item1 = numbers, item2 = text
  set the itemDel to tab
  set the numberFormat to "#.00"  --money
  repeat for each line i in tData
    put (item 1 of i)*1 &tab& item 2 of i &cr after tNewData  --math triggers formatting
  end repeat
  delete char -1 of tNewData
  put "Amount" &tab& "Item" into tHeaders
  set the dgText["true"] of group "DataGrid 1" to tHeaders &cr& tNewData
  set the dgHeaderAlignment["Amount"] of group "DataGrid 1" to "right"  --right justify header
  set the dgColumnAlignment["Amount"] of group "DataGrid 1" to "right"  --right justify column
  send "RefreshList" to group "DataGrid 1"
end mouseUp
```

If a table column's alignment won't follow the Inspector or the dgColumnAlignment property, try this workaround in the column behavior script:

```
on FillInData pData
  set the textAlign of field 1 of me to "right"  --bug 18233 workaround
  set the text of field 1 of me to pData
end FillInData
```

4.13) Color A Line In A Table
To color a line in a data grid table, create a toggle button and customize the table's column behavior script in a button.

This example colors a table's third row red if the row's "line has error" property = true.
Step 1. In button "Toggle Color" put the following code:

```
on mouseUp  --in button, ["line has error"] is built-in property
 local tData
 put the dgDataOfLine[3] of group "DataGrid 1" into tData  --line 3 array data
 put not tData["line has error"] into tData["line has error"]  --toggle true/false
 set the dgDataOfLine[3] of group "DataGrid 1" to tData
end mouseUp
```

Step 2. Color a cell in a table, or color a line in a table.

o To color a cell conditionally, customize that column's column behavior script. In this example, customize the "Amount" column which has numbers. If the "Amount" column's cell is < "10" the number will be colored red. Replace the FillInData handler with the following:

```
on FillInData pData
  set the text of field 1 of me to pData
  if pData < "10" then set the foreColor of field 1 of me to "red"
  else set the foreColor of field 1 of me to empty  --black
end FillInData
```

o To color the entire line 3, create a button for the default behavior of all columns, as done for the TruncateTail example in 4.4) "Override Default Behavior". Add the appropriate portions of handlers below to the custom column behavior script.

```
on FillInData pData
  set the text of field 1 of me to pData
  SetForeColor
end FillInData

setProp dgHilite pBoolean
  if pBoolean then
    set the foreColor of me to the dgProp["hilited text color"] of the dgControl of me
  else SetForeColor
end dgHilite

private command SetForeColor
  if getDataOfIndex(the dgIndex of me, "line has error") then set the textColor of me to "red"
  else set the textColor of me to empty  --black
end SetForeColor
```

4.14) Display Cell Value
To display a table cell value the user clicked, use a mouseDown handler in the table's group script.

```
on mouseDown pMouseBtnNum  --in table's group script
   local tColName, tColValue
   dgMouseDown pMouseBtnNum  --let data grid process mouseDown first
   put the dgColumn of the target into tColName  --column name
   put getDataOfLine(the dgHilitedLine of me, tColName) into tColValue  --cell data clicked
   put "Cell Data Clicked:" && tColValue &cr& "Column Name:" && tColName
   --don't pass mouseDown
end mouseDown

getProp ValueOfSelectedCell  --custom property of this data grid, value of table cell clicked
   put the dgHilitedLines of me into tLine
   return getDataOfLine(tLine, "Col 1")  --value of cell of selected line in specific column
end ValueOfSelectedCell
```

To display a selected table cell value in a specific column, in the script of button "Col 1 Selected Value":

```
on mouseUp
   put "Col 1 Selected Value:" && the ValueOfSelectedCell of grp "DataGrid 1"  --data grid table
end mouseUp
```

==========

CHAPTER 5) HELP WORKING WITH FORMS

Word Wrap, Sort By Key, Rows That Expand, All Rows Expand, Screen Re-Draw, Scroll Selected Row To Top, Display Row Value, Graphic In Row.

5.1) Form Word Wrap - variable line height
To create a data grid form with word wrap, customize the form's row template and row behavior.

Step 1. In the form's Inspector, uncheck Fixed Control Height (off).

Step 2. In the form's Inspector, click the "Row Template" button. Ensure Edit > Select Grouped Controls is checked (on). Select the

field in the upper left "gray" area. In the field's Inspector, uncheck "dontWrap". This will allow the field to word wrap. Save and close.

Step 3. In the form's Inspector, click the "Row Behavior" button. Replace the LayoutControl handler with the following:

```
on LayoutControl pControlRect
  local theFieldRect
  put the rect of field "Label" of me into theFieldRect
  put item 3 of pControlRect - 5 into item 3 of theFieldRect  --new right
  put item 2 of theFieldRect + the formattedHeight of field "Label" of me - 5 into item 4 of theFieldRect  --new bottom
  set the rect of field "Label" of me to theFieldRect  --resize field
  put item 4 of theFieldRect + 4 into item 4 of pControlRect  --update bounding rect of graphic
  set the rect of graphic "Background" of me to pControlRect
end LayoutControl
```

Step 4. In the form Inspector's Contents pane, enter or paste a few lines of text making one line long enough to word wrap. If the text doesn't appear, in message box: send "RefreshList" to group "DataGrid 1"

5.2) Form Sort By Key
To sort the rows of a data grid form by key, use the SortDataByKey command in an option button.

This example uses an option menu with three sort choices: by First Name, Last Name, or Title. The array keys were created when the data was assigned to the form using the dgData property. If the dgText property was used to populate the form, the keys will be "Label 1", "Label 2", ... In the message box:
put line 1 of the dgText["true"] of group "DataGrid 1" --form keys

Put the following code in the option button script:

```
on menuPick pItemName  --in option button
  local tKey, tSortType, tDirection, tCaseSensitive
  switch pItemName
    case "First Name"
      put "Label 1" into tKey
      break
    case "Last Name"
      put "Label 2" into tKey
      break
    case "Title"
      put "Label 3" into tKey
      break
  end switch
  put "text" into tSortType
  put "ascending" into tDirection
  put "false" into tCaseSensitive
  dispatch "SortDataByKey" to group "DataGrid 1" with tKey, tSortType, tDirection, tCaseSensitive
end menuPick
```

5.3) Form Rows That Expand
To make a data grid form with rows that expand and contract when clicking a toggle arrow, the row template layout and row behavior script must be customized. By default all the rows are contracted and only show names. The toggle arrow expands the row to show hidden data. This is a complex build.

Step 1. In the form's Inspector, uncheck Fixed Control Height (off). Or in message box:

set the dgProp["fixed row height"] of group "DataGrid 1" to "false" --off

Step 2. Click the "Row Template" button to add a second field. Ensure that Edit > Select Grouped Controls is unchecked (off). Select the "gray" group in the upper left, Object > Edit Group. Select the original field in the upper left and rename "Name". Copy/paste the original field, position the new field directly below, name "Info", and uncheck dontWrap.

Step 3. Select the original field and move its left edge to make room for two arrow images in the upper left of the "gray" area.

Step 4. Development > Images Library > Standard Images, choose the upper right arrow, img id 200946. Click the "Place Image" button. Rename "ArrowContracted", delete the toolTip number, and position in upper left of the "gray" area. Repeat with the arrow image directly below img id 200946, rename "ArrowExpanded", delete the toolTip number, and position directly over the first arrow. Object > Stop Editing.

Step 5. In the form's Inspector, click the "Row Behavior" button. Replace the FillInData, LayoutControl, and ResetData handlers with the following:

```
on FillInData pDataArray
  set the visible of img "ArrowExpanded" of me to pDataArray["label 1"]  --true/false
  set the visible of img "ArrowContracted" of me to not pDataArray["label 1"]  --opposite
  set the visible of field "Info" of me to pDataArray["label 1"]  --show/hide info
  set the text of field "Name" of me to pDataArray["label 2"]  --name
  set the text of field "Info" of me to pDataArray["label 3"]  --info
end FillInData

on LayoutControl pControlRect  --example 3
  local theFieldRect, theRect
  --resize graphic "Background" and fields, position objects with rectangle property: left,top,right,bottom = 1,2,3,4
  put the rect of field "Name" of me into theFieldRect  --field 1
  put item 3 of pControlRect - 5 into item 3 of theFieldRect  --new right
  set the rect of field "Name" of me to theFieldRect  --resize field
  --
  put the rect of field "Info" of me into theFieldRect  --field 2
  put item 3 of pControlRect - 5 into item 3 of theFieldRect  --new right
  put item 2 of theFieldRect + the formattedHeight of field "Info" of me - 5 into item 4 of theFieldRect  --new bottom
  set the rect of field "Info" of me to theFieldRect  --resize field
  set the left of field "Info" of me to the left of field "Name" of me  --align left
  set the top of field "Info" of me to the bottom of field "Name" of me  --align below field 1
  --
  put pControlRect into theRect
  if the visible of field "Info" of me then  --expand
    put the bottom of field "Info" of me + 5 into item 4 of theRect
  else put the bottom of field "Name" of me + 5 into item 4 of theRect  --contract
  --
  set the rect of graphic "Background" of me to theRect  --resize graphic
end LayoutControl

on ResetData
  set the text of field "Name" of me to empty
  set the text of field "Info" of me to empty
end ResetData
----
```

Step 6. To activate the toggle arrows, in the form's Inspector click the "Row Behavior" button and add the following mouseUp handler below the ResetData handler:

```
on mouseUp pMouseBtnNum  --label 1 = true/false
  if pMouseBtnNum = "1" then  --left click
    switch the short name of the target  --toggle arrow clicked
      case "ArrowExpanded"
      case "ArrowContracted"
        SetDataOfIndex the dgIndex of me, "Label 1", the short name of the target = "ArrowContracted"
        RefreshIndex the dgIndex of me  --redraw
        break
    end switch
  end if
end mouseUp
```

Step 7. Add tab delimited line list text to the form Inspector's Contents pane. Item 1 = "true", item 2 = name, item 3 = info. When data is added as text in the Inspector's Contents pane or with the dgText property, the keys automatically created are "Label 1", "Label 2", ... that map to data item 1, item 2,....

```
true    James Smith    This is some info on James. Add enough text to force a word wrap.
true    Amy Wilson     Info on Amy.
true    Lisa Watts     Some information on Lisa.
```

Step 8. Finally, in message box:

```
send "ResetList" to group "DataGrid 1"  --after changes to row template/behavior
send "RefreshList" to group "DataGrid 1"  --after data changes
```

The Result

When the user clicks on a toggle arrow, the form updates the true/false "Label 1" key in the row's data and redraws the row using SetDataOfIndex and RefreshIndex. If the image "ArrowContracted" is visible and clicked, the SetDataOfIndex evaluates to true which in turn sets the row to be in its expanded state, and toggles the state of the row.

5.4) All Rows Expand

To expand or collapse all form rows (allowing word wrapping), use the fixed row height, row template, and dontWrap properties. In a button script:

```
on mouseUp  --expand/contract form rows, in button script
  local tNewSetting, tRowTemplateRef
  lock screen
  put not the dgProp["fixed row height"] of grp "DataGrid 1" into tNewSetting
  set the dgProp["fixed row height"] of grp "DataGrid 1" to tNewSetting  --toggle t/f fixed line height
  put the dgProp["row template"] of grp "DataGrid 1" into tRowTemplateRef
  set the dontWrap of fld "Label" of tRowTemplateRef to tNewSetting  --toggle t/f dontWrap
  set the dgVScroll of grp "DataGrid 1" to 0  --top
  send "ResetList" to grp "DataGrid 1" --update changes to row template
end mouseUp
```

5.5) Speed Up Screen Redraw When "Fixed Row Height" Is False

When the dgProp["fixed row height"] property is set to false, the data grid must draw all records to determine the total height. This means the FillInData and LayoutControl handlers are called for every record in the data grid. This can be time intensive for large data sets.

A technique to speed up the calculation of the total height of the data uses the CalculateFormattedHeight message. When the data grid loops through the data to calculate the height, the CalculateFormattedHeight message is sent to the Row Template. If a Row Behavior handler intercepts this message, the data grid will use the integer value you return rather than calling the FillInData and LayoutControl handlers.

The following example uses a CalculateFormattedHeight handler to return the height of a row based on whether or not the row is expanded. This technique only works if all the expanded rows are about the same height and all the contracted rows are about the same height. Place the following handler in the form's row behavior script:

```
on CalculateFormattedHeight pDataArray
  if pDataArray["label 1"] then return "75"  --true, expanded height
  else return "25"  --contracted height in pixels
end CalculateFormattedHeight
```

5.6) Scroll Selected Row To Top Of Form
To scroll a selected row to the top of a data grid form, use the dgHilitedIndex, dgRectOfIndex (or dgRectOfLine), and dgVScroll properties. This technique is useful when rows are not fixed height.

```
on mouseUp  --in button script
  local tIndex, tControlRect, tGridRect, tOffset
  put the dgHilitedIndex of group "DataGrid 1" into tIndex
  put the dgRectOfIndex[tIndex] of group "DataGrid 1" into tControlRect
  put the rect of group "DataGrid 1" into tGridRect
  put item 2 of tGridRect - item 2 of tControlRect into tOffset
  set the dgVScroll of group "DataGrid 1" to the dgVScroll of group "DataGrid 1" - tOffset
end mouseUp
```

5.7) Display Row Value
To display a form row value the user clicked, use a mouseDown handler in the form's group script.

```
on mouseDown pMouseBtnNum  --in form's group script
   local tRowName, tRowValue
   dgMouseDown pMouseBtnNum  --let data grid process mouseDown first
   put "Label 1" into tRowName  --key name or "Label 1", 2, ... corresponding to item 1, item 2, ...
   put getDataOfLine(the dgHilitedLine of me, tRowName) into tRowValue  --row value clicked
   put "Row Value Clicked:" && tRowValue &cr& "Key Name:" && tRowName
   --don't pass mouseDown
end mouseDown

getProp ValueOfSelectedCell  --custom property of this data grid, value of table cell clicked
   put the dgHilitedLines of me into tLine
   return getDataOfLine(tLine, "Label 1")  --value of row in specific key/label
end ValueOfSelectedCell
```

To display a selected form row value in a specific key/label, in the script of button "Label 1 Selected Value":

```
on mouseUp
   put "Label 1 Selected Value:" && the ValueOfSelectedCell of grp "DataGrid 1"  --data grid form
end mouseUp
```

5.8) Graphic In Row
To display a graphic line in a form row, place the graphic line in the form's row template.

Step 1. Drag a graphic line from the Tools palette. In the graphic's Inspector, name "Divider", adjust the color, dashes, lineSize, width.
Step 2. Copy the graphic. In the form's Inspector, click button "Row Template". Ensure Edit > Select Grouped Controls is unchecked (off).

Step 3. Select the "gray" group in the upper left, Object > Edit Group. Paste the graphic into the group and position the line just below the bottom field. This will be the divider between rows. Save with ctrl-S. Text style, alignment, and color are properties set in the field's Inspector in the row template group.

Step 4. In the form's Inspector, click button "Row Behavior". In the LayoutControl handler, add the graphic "Divider" portion.

```
on LayoutControl pControlRect  --row template layout, rect=L,T,R,B
  local theFieldRect
  ----
  put the rect of fld "Label" of me into theFieldRect  --fld "Label"
  put item 1 of pControlRect into item 1 of theFieldRect  --left
  put item 3 of pControlRect into item 3 of theFieldRect  --right
  set the rect of fld "Label" of me to theFieldRect  --resize
  ----
  put the rect of graphic "Divider" of me into theFieldRect  --graphic "Divider"
  put item 1 of pControlRect + 10 into item 1 of theFieldRect  --left
  put item 3 of pControlRect - 10 into item 3 of theFieldRect  --right
  set the rect of graphic "Divider" of me to theFieldRect --resize
  ----
  set the rect of graphic "Background" of me to pControlRect
end LayoutControl
```

Step 5. In the form's Inspector:
- Uncheck: alternating row colors.
- Check: fixed row height, and manually adjust the empty row height to accomodate text height.
- Colors & Patterns: background and rows must be the same color, selection hilites may be any light color that won't mask text.

Step 6. In message box:
send "ResetList" to group "DataGrid 1" --after changes to row template.
==========

CHAPTER 6) THE BUILT-IN FIELD EDITOR
Table Cell And Form Row Editing, Edit HTMLText And UnicodeText, Select Text In The Field Editor, Save User Changes To External Source, Customize The Field Editor Behavior.

Double-click on table cell text or form row text to open the built-in field editor. The user can edit text directly. ReturnKey closes the field editor, tabKey closes and opens the next table cell's field editor. This action can be turned off in the Inspector (uncheck Allow Editing), or modified in the column/row behavior script and table/form group script.

6.1) Table Cell And Form Row Editing
By default a table and form can be edited if the user double-clicks on text. This behavior can be customized in the column/row behavior script and table/form group script.

To custom edit table cells or form rows, see the Dictionary's data grid section entries for: EditFieldText, EditValue, EditCell, EditCellOfIndex, EditKey, EditKeyOfIndex, and CloseFieldEditor.
---
o EditFieldText. Command to create an editor for a field you specify. The default column/row behavior calls this command with three parameters so that data is automatically saved after the user finishes editing.

o EditValue. Message sent when a request to edit a field's contents has been made. The default column/row behavior calls EditFieldText when this message is received.
---
o EditCell and EditCellOfIndex. Commands that open a table cell for editing. Each takes the name of the column to edit and the line or index to edit.
o EditKey and EditKeyOfIndex. Commands that open a row field for editing. Each takes the name of the key to edit and the line or

index to edit.

Here are two examples for a column/row behavior script:

```
on mouseDown pBtnNum  --example 1, in column/row behavior script
  if pBtnNum = "1" then  --left click
    if the short name of the target = "FirstName" then
      put "FirstName" into tColKey
      put the dgHilitedLine of me into tLineNum
      EditCell tColKey, tLineNum  --table (comment out for form)
      --EditKey tColKey, tLineNum  --form (comment out for table)
    end if
  end if
end mouseDown

on mouseDown pBtnNum  --example 2, in column/row behavior script
  if pBtnNum = "1" then  --left click
    if the short name of the target = "FirstName" then
      put "FirstName" into tColKey
      put the dgHilitedIndex of me into tIndex
      EditCellOfIndex tColKey, tIndex  --table (comment out for form)
      --EditKeyOfIndex tColKey, tIndex  --form (comment out for table)
    end if
  end if
end mouseDown
```

Either of the above calls will trigger the EditValue message. EditValue can be thought of as a central message to open a field for editing text. A handler for EditValue is where you call EditFieldText.

```
on EditValue pKey  --triggered by EditCell, EditKey, EditCellOfIndex, or EditKeyOfIndex
  EditFieldText the long id of me, the dgIndex of me, the dgColumn of me  --table (comment out for form)
  --EditFieldText the long id of field pKey of me, the dgHilitedIndex of me, pKey  --form (comment out for table)
  --(auto-saves when parameter 2 and 3 are included)
end EditValue
```

---

o CloseFieldEditor message is sent to the field targeted in EditFieldText's first parameter if the user makes changes in the field editor. An example of storing a value in dgData/dgText with CloseFieldEditor, required if only parameter 1 was passed with EditFieldText.

```
on CloseFieldEditor pFieldEditor  --in column/row behavior script
  put the dgIndex of me into tIndex
  put the dgDataOfIndex[tIndex] of the dgControl of me into tData
  put the text of pFieldEditor into tData[the dgColumn of me]  --table (comment out for form)
  --put the text of pFieldEditor into tData[the short name of the target]  --form using dgData (comment out for table)
  --put the text of pFieldEditor into tData["Label 1"]  --form using dgText: Label 1,2,... (comment out for table)
  set the dgDataOfIndex[tIndex] of the dgControl of me to tData
end CloseFieldEditor
```

For table: tData[the dgColumn of me]
For form: tData[the short name of the target] --form using dgData to populate, and name of field = key used in array.
For form: tData["Label 1"] --form using dgText to populate, and item number of field = "Label <itemNumber>".

To determine the headers/keys of a data grid, in message box:
put line 1 of the dgText["true"] of group "DataGrid 1" --headers/keys

---

6.2) Edit HTMLText And UnicodeText
The default table behavior when editing cell contents uses the text property of the cell. Override the default behavior in a custom column behavior script.

Text in a table can display styling, like bold or italic, if the formatting information is contained in the data. If the user edits the content of the cell the formatting is lost. This is because the data grid edits using the text property of the field by default.

Change the default value of the field editor with the dgTemplateFieldEditor property. Valid values are text, htmlText, rtfText, utf8Text, and unicodeText. Set the property before calling EditFieldText in a custom column behavior script. For example, to maintain styled text while editing:

```
command EditValue  --in table's column behavior script
  set the dgTemplateFieldEditor["htmlText"] of the dgControl of me to the htmlText of me  --maintain styling
  EditFieldText the long id of me, the dgIndex of me, the dgColumn of me
end EditValue
```

6.3) Select Text In The Field Editor When It Opens
The default table behavior when editing cell contents is to put the cursor at the end of the field. Override the default behavior in a custom column behavior script.

To select all the cell text when editing cell contents, use the dgTemplateFieldEditor property. The valid value is select text. Set the property before calling EditFieldText in a custom column behavior script. For example, to select all cell contents when the field editor opens:

```
command EditValue  --in table's column behavior script
  set the dgTemplateFieldEditor["select text"] of the dgControl of me to "true"  --select all cell contents
  EditFieldText the long id of me, the dgIndex of me, the dgColumn of me
end EditValue
```

6.4) Save User Changes To External Source
o Call EditFieldText With 3 Parameters (Simpler)
When calling EditFieldText with all three parameters (by default), text entered is automatically saved in the dgData array. Save the text of the field editor to a backup external source with a CloseFieldEditor handler in the table's group script. If saving anything other than the text of pFieldEditor, the data source will not match the dgData value. Two examples to backup entered text:

```
on CloseFieldEditor pFieldEditor  --example 1, in table's group script
  local tColEdited, tNewText, tTable, tRowID
  put the dgColumn of the target into tColEdited
  put the text of pFieldEditor into tNewText
  put "Person" into tTable  --database table
  put GetDataOfIndex(the dgIndex of the target, "id") into tRowID  --database row id
  SaveDataToDatabase tTable, tRowID, tColEdited, tNewText  --custom command to update database
end CloseFieldEditor

on CloseFieldEditor pFieldEditor  --example 2, in table's group script
   local tColEdited, tLineNum, tNewText
   put the dgColumn of the target into tColEdited
   put the dgHilitedLine of me into tLineNum
   put the text of pFieldEditor into tNewText
   put tColEdited &","& tLineNum &","& tNewText  --into msg box
end CloseFieldEditor
```

---
o Call EditFieldText With 1 Parameter (More Flexible)
When calling EditFieldText with only parameter one, text entered is not automatically saved in the dgData array. You are responsible for saving changes to the dgData, then making a backup to an external source. An example to backup then save entered text:

```
on CloseFieldEditor pFieldEditor  --in table's group script
  local tColEdited, tNewText, tTable, tRowID, tData
  put the dgColumn of the target into tColEdited
  put uniDecode(the unicodeText of pFieldEditor, "utf8") into tNewText
  put "Person" into tTable  --database table
  put GetDataOfIndex(the dgIndex of the target, "id") into tRowID  --database row id
  SaveDataToDatabase tTable, tRowID, tColEdited, tNewText  --custom command to update database
  --
  put the dgDataOfIndex[the dgIndex of the target] of me into tData
  put tNewText into tData[tColEdited]
  set the dgDataOfIndex[the dgIndex of the target] of me to tData  --save & refresh display
end CloseFieldEditor
```

6.5) Customize The Field Editor Behavior
By default the data grid field editor allows users to enter data and save it back to the data grid. To make data entry behave differently, assign your own behavior button script to the field editor before it opens.

Step 1. Create a button "MyFieldBehavior" next to a data grid table to hold the behavior script that will modify the field editor.

Step 2. Begin with the default behavior script. In message box:
set the script of button "MyFieldBehavior" to the script of button "Field Editor" of stack "revDataGridLibrary"

Step 3. In your behavior button script, make any customizations.

Step 4. When a data grid displays the field editor (the user double-clicks on a cell in a table) a preOpenFieldEditor message is sent to the data grid's group script. The first parameter is a reference to the field editor control. This is where a behavior script can be assigned to the field.

```
on preOpenFieldEditor pFieldEditor  --in table's group script
  set the behavior of pFieldEditor to the long id of button "MyFieldBehavior"
end preOpenFieldEditor
```
==========

CHAPTER 7) STANDALONES WITH A DATA GRID
Deploy Standalone With A Data Grid.

7.1) Deploy A Standalone With A Data Grid
A data grid relies on the stack "revDataGridLibrary" in order to function properly. The standalone builder detects the presence of a data grid and automatically adds the required library to the application package.

Launcher Stacks or Splash Stacks
Some developers prefer to use a Launcher (Splash) stack technique. This technique builds a standalone using a stack with very little code in it that opens the main stack that contains the data grid. In this case the standalone builder would not detect a data grid. You must tell the builder you require the inclusion of the data grid library. This will ensure any stack loaded by the standalone can use data grids.

Option 1: Search For Inclusions. Include a substack named "Data Grid Templates Decoy" (exact spelling) as part of the Splash stack if your working stack includes a data grid. This will force the auto-inclusions routine to include the data grid library in the

standalone build.

Option 2: Select Inclusions. File > Standalone Application Settings, General tab, select "Select Inclusions", hilite "Data Grid". Any other inclusions necessary for the main stack must also be selected.
==========

CHAPTER 8) ADVANCED OPTIONS
What Not To Do, Create Data Grid By Code, Create Table By Code, Display Large Data, Display SQLite Database In Table.

8.1) What Not To Do
o Don't call a handler that redraws the data grid from within a control in the data grid.
This will generate an error if deleting a control that is currently executing code. Avoid calling a handler that refreshes the data grid from within a control by using send in time or placing the code in the data grid group script.

```
send "DeleteIndex theIndex" to the dgControl of me in 0 seconds  --in row behavior script

on mouseUp pMouseBtnNum  --in group script, delete hilited index
  if pMouseBtnNum = "1" then  --left click
    put the dgHilitedIndex of me into tIndex
    DeleteIndex tIndex
  end if
end mouseUp
```
---
o Don't draw a data grid on a card not open.
When a data grid renders, it dynamically creates fields and accesses certain properties. Some of these properties can not be properly reported by the Engine unless the field is on an open card.

o Don't lock messages when accessing data grid properties.
If messages are locked when accessing a data grid property, the correct value will not be returned/set. A data grid relies on getProp/setProp handlers to function. When messages are locked these are not triggered.

o Don't password protect the "Data Grid Templates" stack.
The data grid copies the templates from the "Data Grid Templates" stack. If you password protect this stack the data grid will be unable to copy the templates.

o Don't rename the "Data Grid Templates" stack.
Renaming this stack will cause all data grids with templates stored in the stack to stop working. Since a data grid can no longer locate their custom templates they will fail to draw properly.

o Don't search when data is being loaded from an external source.

o Stop editing the column/row template group before drawing the data grid.
When editing a group, the Engine no longer knows that the group exists and the data grid will fail to draw. Choose Object > Stop Editing or click the Stop Editing icon on the Toolbar.

8.2) Create A Data Grid By Code
Step 1. Copy data grid from revDataGridLibrary.
The data grid template is stored in the "revDataGridLibrary" stack and can be copied using some code similar to this:

```
copy group "DataGrid" of group "Templates" of stack "revDataGridLibrary" to card "MyCard" of stack "MyStack"
put it into tDataGridRef
```

Step 2. Set the "style" property.

```
set the dgProp["style"] of tDataGridRef to "table"  --table / form
```

Step 3. Assign a row template. Create this ahead of time using the IDE. For example, create a data grid and then delete it while leaving the row template behind (it will exist on a card in "Data Grid Templates xxx" stack).
set the dgProp["row template"] of tDataGridRef to the long id of group "MyRowTemplate" of stack "MyStack"

8.3) Create Table By Code
Step 1. Prepare the tab delimited data.
The first row of the data should include the header names of the columns, each name unique, no empty names allowed. One way to create tab delimited data is with a spreadsheet. Choose "Save as..." and select "text file (tab delimited)". To load the data from such a file:

```
on mouseUp  --in button script
  answer file "Please choose a tab delimited text file to import..."
  if it = empty then exit to top
  put url ("file:" & it) into tData
  put tData into field "Table Data"  --table field
end mouseUp
```

Step 2. Create the data grid table.
Each data grid consists of deeply nested groups with scripts and objects. Luckily, you do not need to recreate all that from scratch. Instead, copy the template data grid from the data grid library. Because this library is necessary for all data grid functionality, it will be added to standalones by the IDE
automatically.

Step 3. Create the row template.
In theory, the data grid is ready for use. But for the IDE to interact correctly with the new data grid, it needs a substack with the row template, just as the IDE would. The second card of that substack needs a group called "Row Template", and the data grid needs to point to that group.

Step 4. Fill in the data.
As a last step, insert the prepared data into the data grid table. This assumes the first row of data contains unique header names for each column. Dynamically create all necessary columns with the dgProp["columns"] property. The names must be in a return delimited list.

Put Everything Together
Copy the following mouseUp handler into the button script of an empty stack.

```
on mouseUp  --in button script of empty stack
  local tData, tMainStack, tName, tColumns
  put field 1 into tData  --tab delimited, line 1 = header names, or open text file:
  --answer file "Please select a tab delimited text file." with type "any file" or type "tab file|tab|TEXT" or type "txt file|txt|TEXT"
  --if it = empty then exit mouseUp
  --put url ("file:" & it) into tData
  lock screen  --faster
  if there is a group "MyDataGrid" then delete group "MyDataGrid"
  --
  copy group "DataGrid" of group "Templates" of stack "revDataGridLibrary" to this card
  set the name of it to "MyDataGrid"
  set the dgProp["style"] of group "MyDataGrid" to "table"
  set the rectangle of group "MyDataGrid" to 0, the bottom of me + 10, the width of this card *.5, the height of this card *.5
  --
  put the name of this stack into tMainStack
  put "Data Grid Templates" && the seconds into tName
  create invisible stack tName
  set the mainStack of stack tName to tMainStack
```

```
  go stack tName
  create card
  create field "label"
  create graphic "Background"
  group field 1 and graphic 1
  set the name of last group to "Row Template"
  --
  go stack tMainStack
  set the dgProp["Row Template"] of group "MyDataGrid" to the long id of group "Row Template" of stack tName
  put line 1 of tData into tColumns  --column headers
  replace tab&tab with tab& " " &tab in tColumns  --insert space if empty column name
  replace tab with return in tColumns  --headers in return delimited list
  set the dgProp["columns"] of group "MyDataGrid" to tColumns  --must populate headers first
  set the dgText["true"] of group "MyDataGrid" to tData  --headers in line 1
end mouseUp
```

8.4) Displaying Large Amounts Of Data

Setting the dgText property or creating an array and setting the dgData property of a data grid is the easiest way to display data in a data grid. But what about situations where creating an array is too time intensive and the data is already in another format? A data grid can handle these situations with Callbacks.

The dgNumberOfRecords Property

Normally a data grid reports the number of records based on the number of numeric indexes in the first dimension of the dgData array. If you set the dgNumberOfRecords property, however, the data grid stops using an internal array and issues a Callback message whenever it needs to display data in a line.

Important: When using this technique, properties like dgData, dgText, dgDataOfIndex, etc. will no longer return values. The data grid is just displaying records from your data source. It does not store any of that data internally.

GetDataForLine Callback

Set the dgNumberOfRecords to 10000 and the data grid would start sending the GetDataForLine message whenever it needed to display a row. Your responsibility is to fill in the data that the data grid needs by handling the GetDataForLine message. To do that, define GetDataForLine as follows:

```
command GetDataForLine pLine, @pData  --in group script
end GetDataForLine
```

Define this handler in the data grid group script or anywhere else in the message path. Fill in pData with the appropriate data and the data grid will display it.

8.5) Display SQLite Database In Table

To display a SQLite database in a data grid table, extract tab delimited data from the database using revDataFromQuery, and display the data using dgText.

Step 1. In the stack that generated the SQLite database there would usually be a display field of tab delimited data with an option to extract all data. If there is only a SQLite database file, the data will have to be extracted in a stack.

The User Guide Chapter 12.4 has a SQLite example to use. In stack "SQLite1" put the following code in the stack script. It creates the database or opens an existing one, and loads it into a field:

```
local sDatabaseID  --script local, available to handlers below, persists until quit
on preOpenStack  --in stack script
  local tName, tDatabaseFile, tTableSQL, tSQLStatement
```

```
  put the short name of this stack into tName
  put specialFolderPath("documents") & "/" & tName & ".sqlite" into tDatabaseFile
  if there is not a file tDatabaseFile then  --create db file
    put revOpenDatabase("sqlite",tDatabaseFile) into sDatabaseID
    put "CREATE TABLE books(BookID INTEGER PRIMARY KEY AUTOINCREMENT, Author varchar(255), Title varchar(255))" into
tTableSQL
    revExecuteSQL sDatabaseID,tTableSQL
    --add some initial entries:
    put "INSERT into books(Author,Title) VALUES ('Tolkien J','Lord Of The Rings')" into tSQLStatement
    revExecuteSQL sDatabaseID,tSQLStatement
    put "INSERT into books(Author,Title) VALUES ('Austen J','Pride And Prejudice')" into tSQLStatement
    revExecuteSQL sDatabaseID,tSQLStatement
    put "INSERT into books(Author,Title) VALUES ('Baum F','Wizard Of Oz')" into tSQLStatement
    revExecuteSQL sDatabaseID,tSQLStatement
  else put revOpenDatabase("sqlite",tDatabaseFile) into sDatabaseID  --db file exists
  put dbListAll() into fld "Display"
end preOpenStack

function dbListAll  --retrieve all from database
  local tSQLStatement
  put "SELECT * FROM books" into tSQLStatement  --select all
  return revDataFromQuery(tab,cr,sDatabaseID,tSQLStatement)
end dbListAll
```

Step 2. To see the database, place scrolling field "Display" on the stack. In message box:

```
set the tabStops of fld "Display" to "90"  --for tabbed data
```

Step 3. Save the stack with ctrl-S. Close the stack and re-open to trigger the preOpenStack handler. A .sqlite database is created in the Documents folder.

Step 4. Drag a data grid from the Tools palette. By default, it is a data grid table. In the script of button "SQLite To DGTable", put the following code:

```
on mouseUp  --in button script, populate dg table
  local tData, tHeaders
  put field "Display" into tData  --tab delimited line list
  put "ID" &cr& "Author" &cr& "Title" into tHeaders
  set the dgProp["columns"] of group "DataGrid 1" to tHeaders  --populate headers first, line list
  replace cr with tab in tHeaders  --tab delimited
 set the dgText["true"] of group "DataGrid 1" to tHeaders &cr& tData  --headers in line 1
end mouseUp
```

If given only the SQLite database file, to determine the table name and column names, use the revDatabaseTableNames and revDatabaseColumnNames functions. With this information, the database can then be displayed. In an empty stack place in the script of button "SQLite Info":

```
on mouseUp  --in button script, get database info
  local tDatabaseFile, tDatabaseID, tTables, tFirstTable
  answer file "Please select a SQLite database file." with type "SQLite|sqlite" titled "SQLite Database"
  if it = empty then exit to top
  put it into tDatabaseFile
  put revOpenDatabase("sqlite",tDatabaseFile) into tDatabaseID  --db file exists
  put revDatabaseTableNames(tDatabaseID) into tTables
  put line 1 of tTables into tFirstTable
  put "Table Names:" && tTables &cr&cr& "Field Names For" && tFirstTable &":" && revDatabaseColumnNames(tDatabaseID,
```

tFirstTable)
end mouseUp
==========

APPENDIX A) DATA GRID REFERENCE

Frequently Used Code, General Properties, Table Properties, Table Column Properties, Table Header Properties, Custom Properties, Commands, Functions, Messages Sent, Customized Template Messages Sent, Customized Template Custom Properties

A.1) Frequently Used Code

```
set the dgText["true"] of grp "DataGrid 1" to tData  --tab delimited line list
set the dgData["true"] of grp "DataGrid 1" to tArrayData  --array
send "RefreshList" to grp "DataGrid 1"  --after data changes
send "ResetList" to grp "DataGrid 1"  --after template/behavior changes
set the dgText of grp "DataGrid 1" to empty  --delete contents
set the dgHeaderAlignment["Amount"] of grp "DataGrid 1" to "right"  --align table header
put the dgProp["Columns"] of grp "DataGrid 1"  --header line list
set the backgroundBehavior of grp "DataGrid 1" to "true"  --unpopulated dg on new cds
put the dgNumberOfLines of grp "DataGrid 1"  --line count
```

A.2) General Properties

Get/put/set the general properties of a data grid. See the data grid section of Dictionary.

```
put the dgProp["PropertyName"] of group "DataGrid 1"
```

allow editing, alternate row color, alternate row colors, auto hilite, background color, cache controls, column divider color, column margins, control type, dim on focusOut, dimmed hilite color, fixed row height, hilite color, hilited text color, multiple lines, opaque, persistent data, row color, row height, row template, scroll when hscrollbar is hidden, scroll when vscrollbar is hidden, scrollbar corner offset, show vscrollbar, show hscrollbar, scrollbar width, style, text color, text font, text size, text style.

A.3) Table Properties

Get/put/set the properties of a data grid table.

```
put the dgProp["<PropertyName>"] of group "DataGrid 1"
```

allow column resizing, column divider color, column alignments, column visibility, column widths, columns, column labels, corner color, default column behavior, default header behavior, header background color, header background hilite color, header height, header margins, header text color, header text font, header text size, header text style, show column dividers, show header, sort by column, visible columns.

A.4) Table Column Properties

Get/put/set the properties of individual columns of a data grid table.

```
put the dgColumnSortType["<ColumnName>"] of group "DataGrid 1"
```

dgColumnAlignment, dgColumnIsEditable, dgColumnIsVisible, dgColumnIsResizable, dgColumnLabel, dgColumnMaxWidth, dgColumnMinWidth, dgColumnName, dgColumnTemplate, dgColumnTooltip, dgHeaderTemplate, dgColumnSortDirection, dgColumnSortIsCaseSensitive, dgColumnSortType, dgColumnWidth, dgHeaderAlignment.

A.5) Table Header Properties

Use in a mouseDown/mouseUp handler.

o dgHeader. Returns the long id of the group that contains the controls for the table header. Use to determine if the user clicked a table header.

```
if the dgHeader of the target is not empty then...  --user clicked a table header
```

o dgHeaderControl. Returns the long id of the group that contains the controls for a column header. Use to determine if the user clicked a column header.

if the dgHeaderControl of the target is not empty then... --user clicked a column header
---

A.6) Data Grid Custom Properties
The custom properties of a data grid.
Get/put/set the <CustomProperty> of group "DataGrid 1"
dgControl, dgData, dgDataControlOfIndex, dgDataOfIndex, dgDataOfLine, dgFocus, dgFormattedHeight, dgFormattedWidth, dgNumberOfLines, dgNumberOfRecords, dgText, dgHilitedIndexes, dgHilitedIndex, dgHilitedLines, dgHilitedLine, dgHScroll, dgHScrollPercent, dgIndexes, dgIndexOfLine, dgVScroll, dgVScrollPercent, dgVisibleLines
---

A.7) Data Grid Commands
The commands that can be sent to a data grid, using dispatch or send commands.
put "value" into tArray["property"]
dispatch "AddData" to group "DataGrid 1" with tArray --using dispatch
send "AddData tArray" to group "DataGrid 1" --using send

AddData, AddLine, DeleteIndex, DeleteIndexes, DeleteLine, DeleteLines, EditCell, EditCellOfIndex, EditFieldText, EditKey, EditKeyOfIndex, FindIndex, FindLine, RefreshIndex, RefreshLine, ScrollIndexIntoView, ScrollLineIntoView, SelectAll, SetDataOfIndex, SetDataOfLine, SortByColumn, SortDataByKey, RefreshList, ResetControl, ResetList, ResizeToFit
---

A.8) Data Grid Functions
The functions that return info from a data grid.
o columnControlOfIndex(pColumnName, pIndex). Table only, returns the control for the column of index pIndex in the data grid.
o getDataOfIndex(pIndex, pKey). Returns the internal array for key pIndex.
o getDataOfLine(pLine, pKey). Returns the internal array associated with line pLine.

A.9) Data Grid Messages Sent
o selectionChanged pHilitedIndex, pPrevHilitedIndex. Sent when the user changes the selection.
o editValue pKey. Sent to a table column control when EditCell or EditCellOfIndex is called.

A.10) Customized Template Messages Sent
o FillInData pData. Sent when moving data into the controls for a row or column.
o LayoutControl pControlRect. Sent when it is time to position all of the controls.

A.11) Customized Template Custom Properties
dgLine, dgIndex, dgColumn, dgColumnNumber, dgDataControl (getProp), dgHilite pBoolean (setProp)
==========