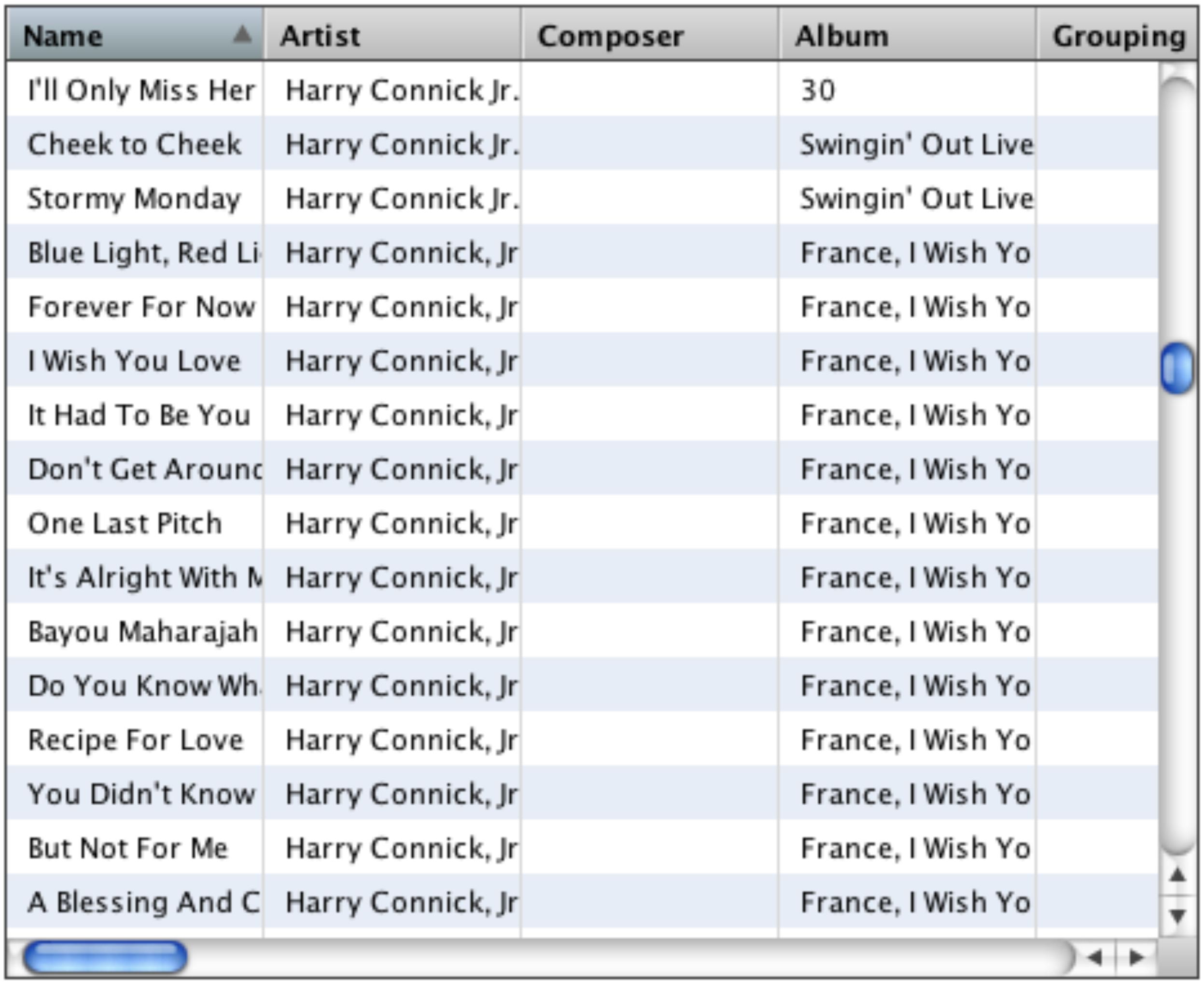
# LiveCode Data Grid

# Introduction

## What is the Data Grid?

The Revolution Data Grid enables you to integrate powerful tables and forms into your Revolution projects. Data grids combine Revolution groups and behaviors to provide you with a simple, yet flexible means of displaying your data in just about any way you want.

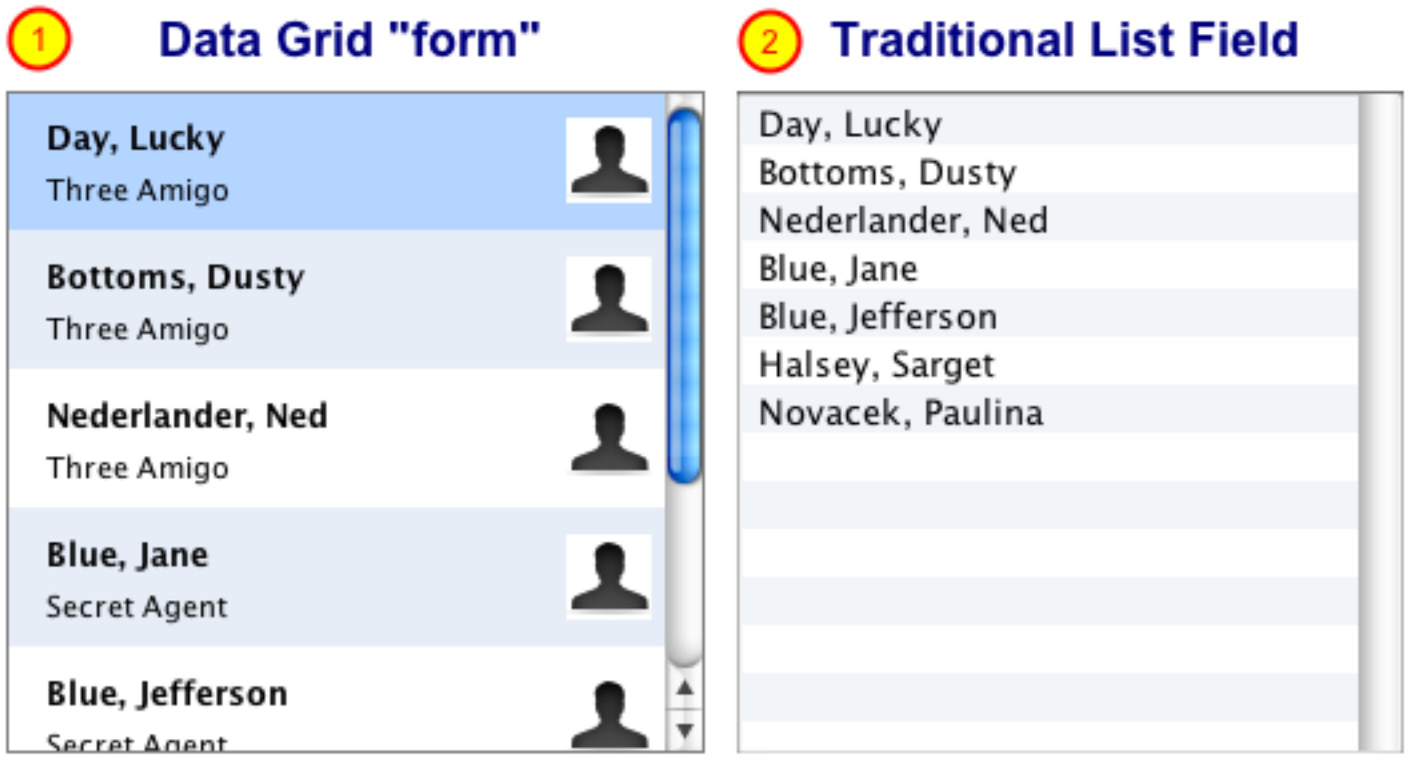
### A Data Grid Table



| Name | Artist | Composer | Album | Grouping |
|---|---|---|---|---|
| I'll Only Miss Her | Harry Connick Jr. | | 30 | |
| Cheek to Cheek | Harry Connick Jr. | | Swingin' Out Live | |
| Stormy Monday | Harry Connick Jr. | | Swingin' Out Live | |
| Blue Light, Red Li | Harry Connick, Jr | | France, I Wish Yo | |
| Forever For Now | Harry Connick, Jr | | France, I Wish Yo | |
| I Wish You Love | Harry Connick, Jr | | France, I Wish Yo | |
| It Had To Be You | Harry Connick, Jr | | France, I Wish Yo | |
| Don't Get Arounc | Harry Connick, Jr | | France, I Wish Yo | |
| One Last Pitch | Harry Connick, Jr | | France, I Wish Yo | |
| It's Alright With M | Harry Connick, Jr | | France, I Wish Yo | |
| Bayou Maharajah | Harry Connick, Jr | | France, I Wish Yo | |
| Do You Know Wh. | Harry Connick, Jr | | France, I Wish Yo | |
| Recipe For Love | Harry Connick, Jr | | France, I Wish Yo | |
| You Didn't Know | Harry Connick, Jr | | France, I Wish Yo | |
| But Not For Me | Harry Connick, Jr | | France, I Wish Yo | |
| A Blessing And C | Harry Connick, Jr | | France, I Wish Yo | |

The *table* style of a data grid allows you to display your data in a modern looking table complete with headers, sorting, column alignment and more. By default a data grid table will use a Revolution field to each column of each row. If you need to customize a column with graphics you can define your own templates for each column. A custom template can be any Revolution control such as a graphic or group.

## A Data Grid Form



The *form* style of a data grid (1) is similar to a list field (2) in Revolution in that it allows you to display records from a data source (text file, database, XML, etc.) that the user can select.

The difference is that the data grid uses a group as a template for each record. This means that for each record you display you can create rich user interfaces using menus, images, etc.

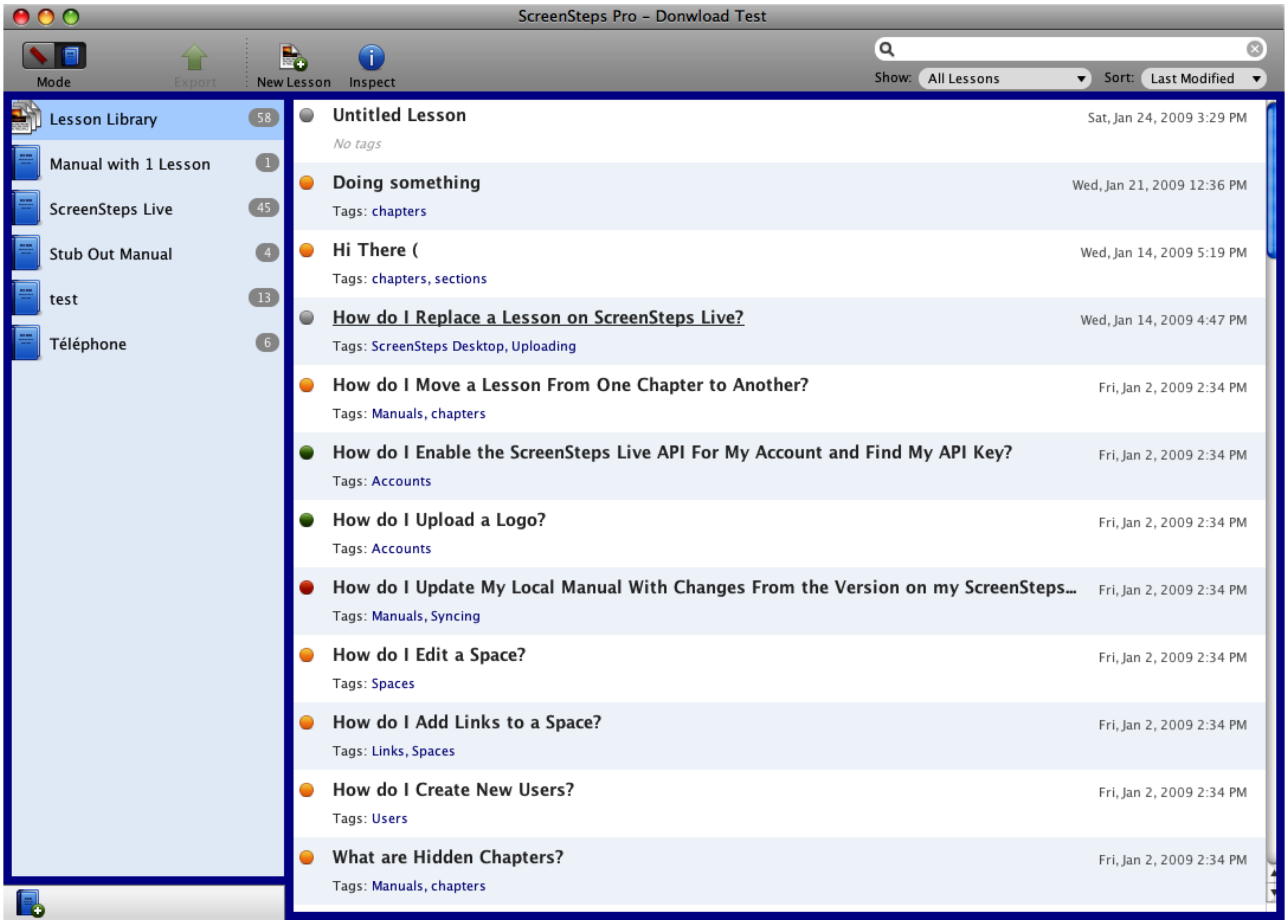
The data grid form is also very fast as it only draws the records that are currently visible on the screen. This means you can display large records sets without slow performance.

Because the data grid form is so easy to use it is not only limited to displaying lots of records. Any list where you want a rich UI and complete control over layout and processing of engine events can benefit from being displayed using a data grid form.

## Can You Show Me Some Examples?

Of course we can! The areas of the screenshots highlighted in blue are instances of the data grid control.

### Data Grid Forms


ScreenSteps Pro – Donwload Test

Lesson Library 58
Manual with 1 Lesson 1
ScreenSteps Live 45
Stub Out Manual 4
test 13
Téléphone 6

Untitled Lesson — Sat, Jan 24, 2009 3:29 PM
No tags

Doing something — Wed, Jan 21, 2009 12:36 PM
Tags: chapters

Hi There ( — Wed, Jan 14, 2009 5:19 PM
Tags: chapters, sections

How do I Replace a Lesson on ScreenSteps Live? — Wed, Jan 14, 2009 4:47 PM
Tags: ScreenSteps Desktop, Uploading

How do I Move a Lesson From One Chapter to Another? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, chapters

How do I Enable the ScreenSteps Live API For My Account and Find My API Key? — Fri, Jan 2, 2009 2:34 PM
Tags: Accounts

How do I Upload a Logo? — Fri, Jan 2, 2009 2:34 PM
Tags: Accounts

How do I Update My Local Manual With Changes From the Version on my ScreenSteps... — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, Syncing

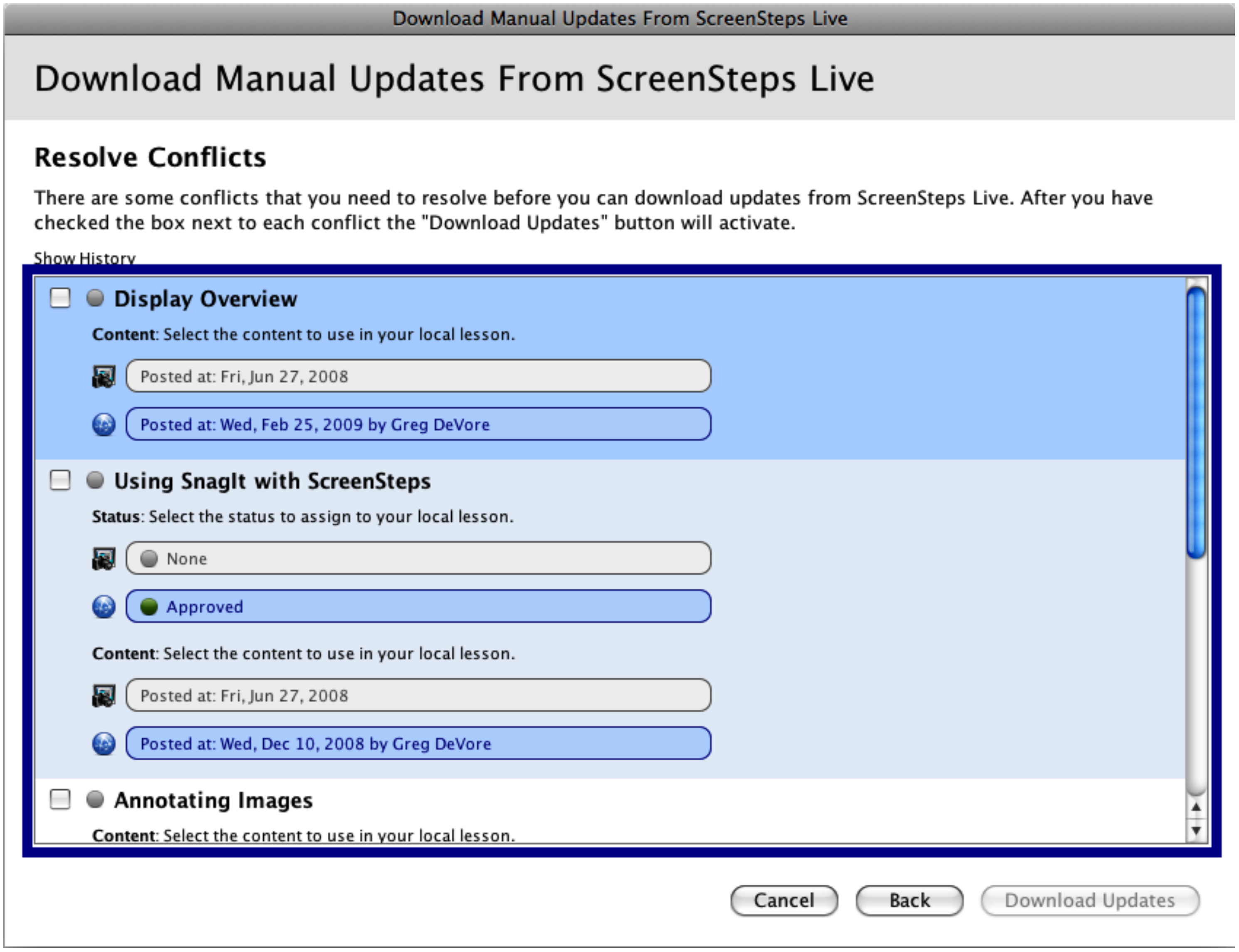
How do I Edit a Space? — Fri, Jan 2, 2009 2:34 PM
Tags: Spaces

How do I Add Links to a Space? — Fri, Jan 2, 2009 2:34 PM
Tags: Links, Spaces

How do I Create New Users? — Fri, Jan 2, 2009 2:34 PM
Tags: Users

What are Hidden Chapters? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, chapters

## Data Grid Form



Download Manual Updates From ScreenSteps Live

Download Manual Updates From ScreenSteps Live

Resolve Conflicts

There are some conflicts that you need to resolve before you can download updates from ScreenSteps Live. After you have checked the box next to each conflict the "Download Updates" button will activate.

Show History

☐ Display Overview

**Content**: Select the content to use in your local lesson.

Posted at: Fri, Jun 27, 2008

Posted at: Wed, Feb 25, 2009 by Greg DeVore

☐ Using SnagIt with ScreenSteps

**Status**: Select the status to assign to your local lesson.

None

Approved

**Content**: Select the content to use in your local lesson.

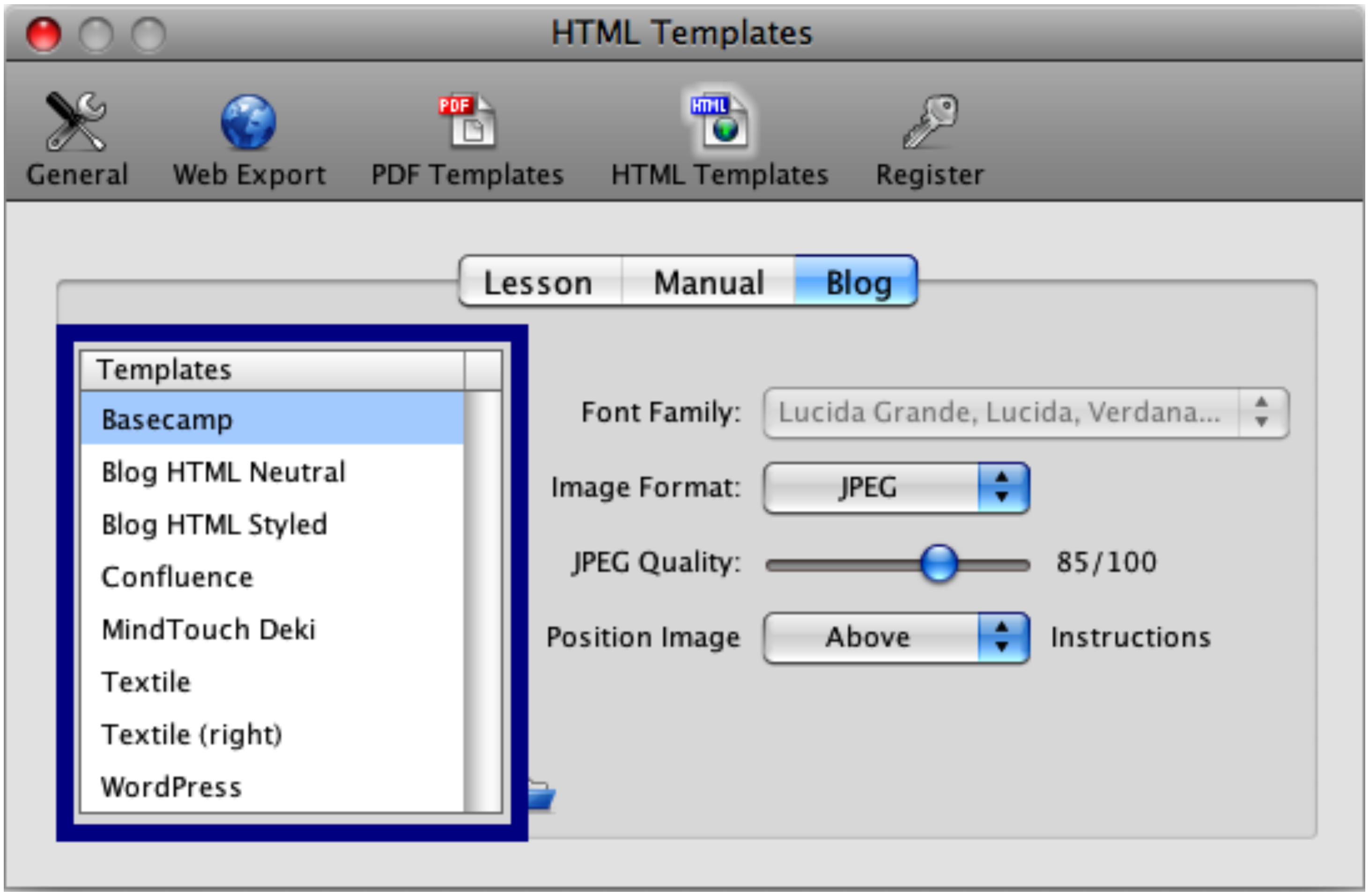
Posted at: Fri, Jun 27, 2008

Posted at: Wed, Dec 10, 2008 by Greg DeVore

☐ Annotating Images

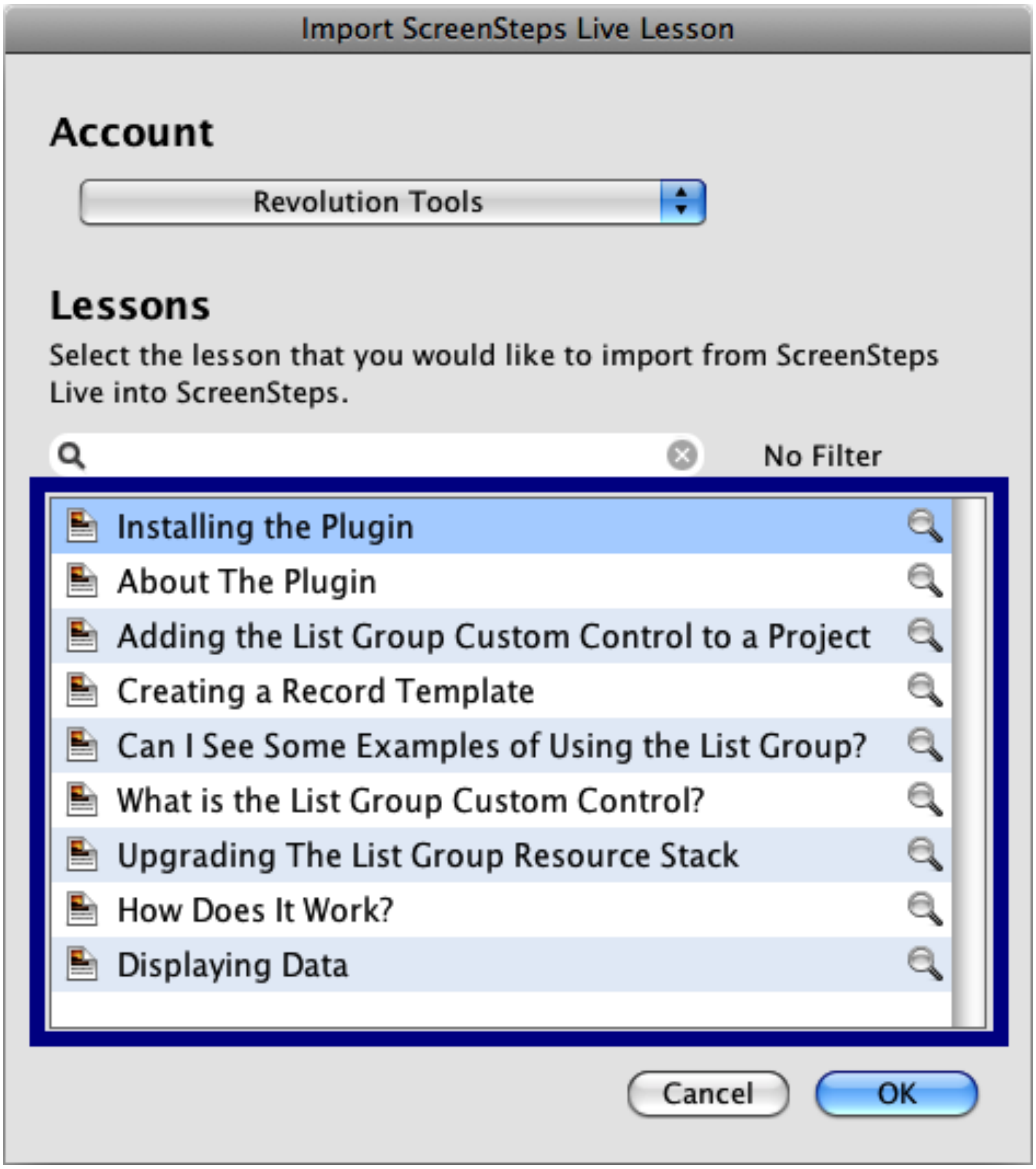
**Content**: Select the content to use in your local lesson.

## Data Grid Table



| Templates |
|---|
| Basecamp |
| Blog HTML Neutral |
| Blog HTML Styled |
| Confluence |
| MindTouch Deki |
| Textile |
| Textile (right) |
| WordPress |

Font Family: Lucida Grande, Lucida, Verdana...
Image Format: JPEG
JPEG Quality: 85/100
Position Image Above Instructions



Import ScreenSteps Live Lesson

**Account**

Revolution Tools

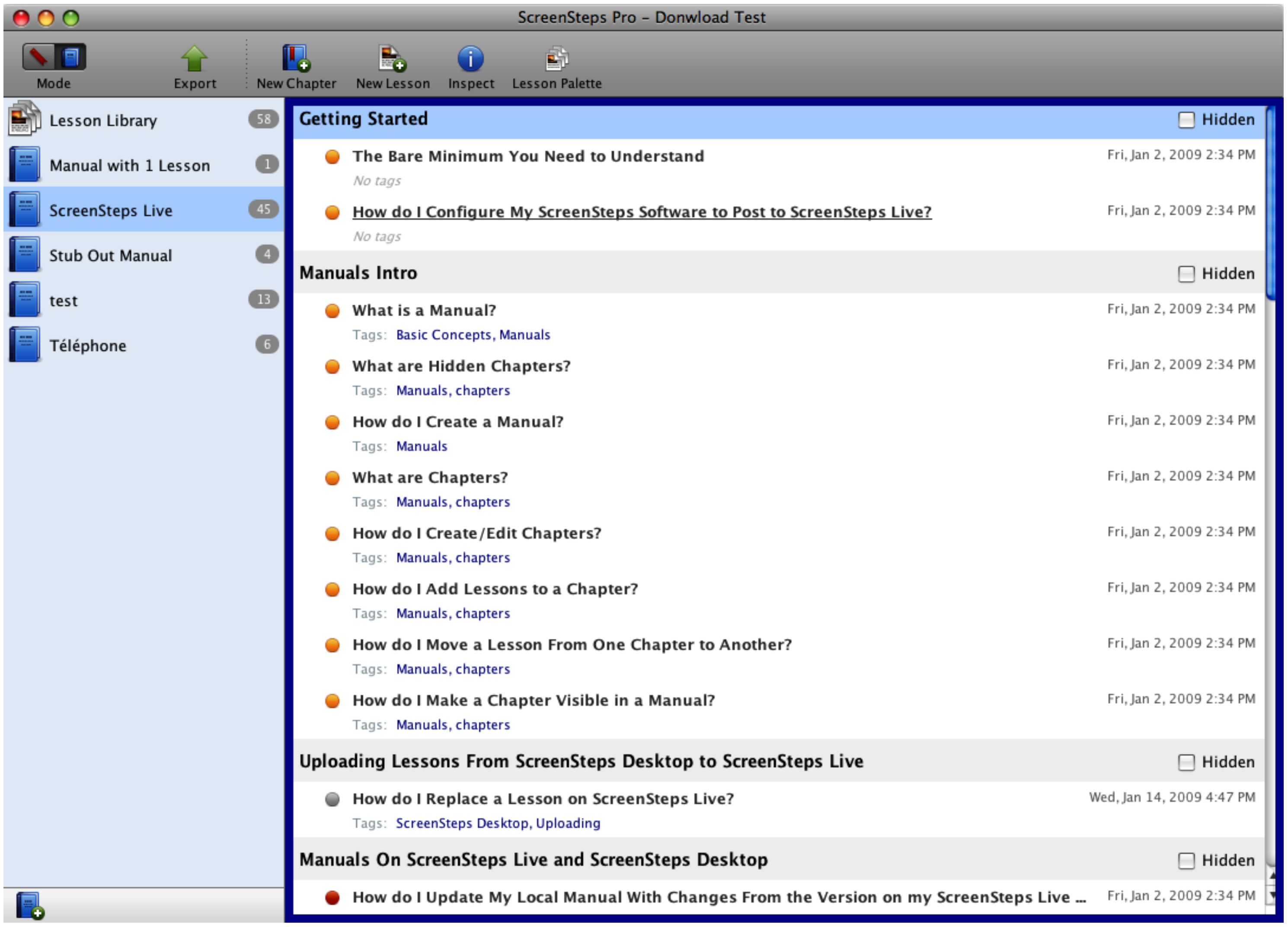
**Lessons**

Select the lesson that you would like to import from ScreenSteps Live into ScreenSteps.

No Filter

- Installing the Plugin
- About The Plugin
- Adding the List Group Custom Control to a Project
- Creating a Record Template
- Can I See Some Examples of Using the List Group?
- What is the List Group Custom Control?
- Upgrading The List Group Resource Stack
- How Does It Work?
- Displaying Data

## Data Grid Form



Lesson Library 58
Manual with 1 Lesson 1
ScreenSteps Live 45
Stub Out Manual 4
test 13
Téléphone 6

**Getting Started** ☐ Hidden

● The Bare Minimum You Need to Understand — Fri, Jan 2, 2009 2:34 PM
*No tags*

● How do I Configure My ScreenSteps Software to Post to ScreenSteps Live? — Fri, Jan 2, 2009 2:34 PM
*No tags*

**Manuals Intro** ☐ Hidden

● What is a Manual? — Fri, Jan 2, 2009 2:34 PM
Tags: Basic Concepts, Manuals

● What are Hidden Chapters? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, chapters

● How do I Create a Manual? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals

● What are Chapters? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, chapters

● How do I Create/Edit Chapters? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, chapters

● How do I Add Lessons to a Chapter? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, chapters

● How do I Move a Lesson From One Chapter to Another? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, chapters

● How do I Make a Chapter Visible in a Manual? — Fri, Jan 2, 2009 2:34 PM
Tags: Manuals, chapters

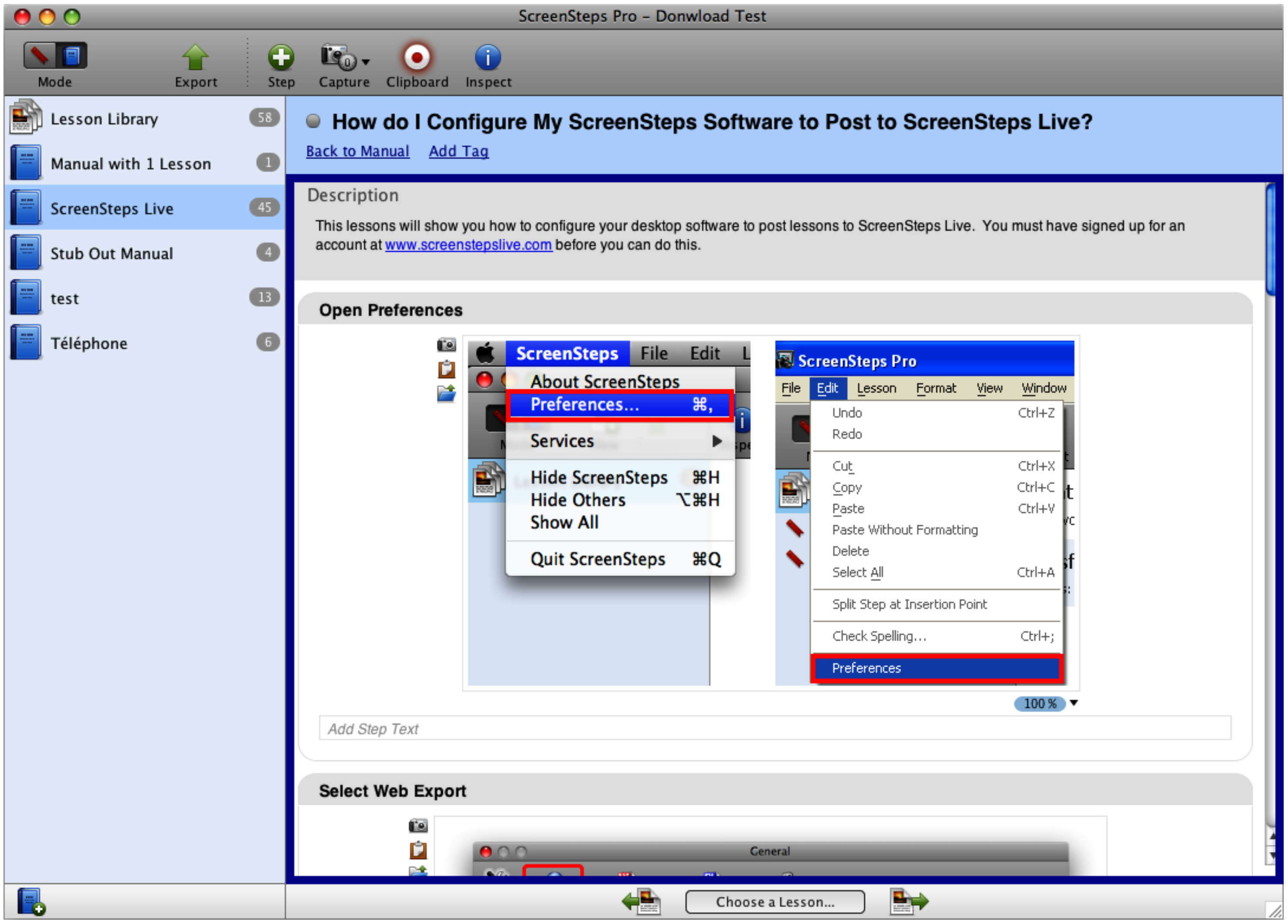
**Uploading Lessons From ScreenSteps Desktop to ScreenSteps Live** ☐ Hidden

● How do I Replace a Lesson on ScreenSteps Live? — Wed, Jan 14, 2009 4:47 PM
Tags: ScreenSteps Desktop, Uploading

**Manuals On ScreenSteps Live and ScreenSteps Desktop** ☐ Hidden

● How do I Update My Local Manual With Changes From the Version on my ScreenSteps Live ... — Fri, Jan 2, 2009 2:34 PM

## Data Grid Form

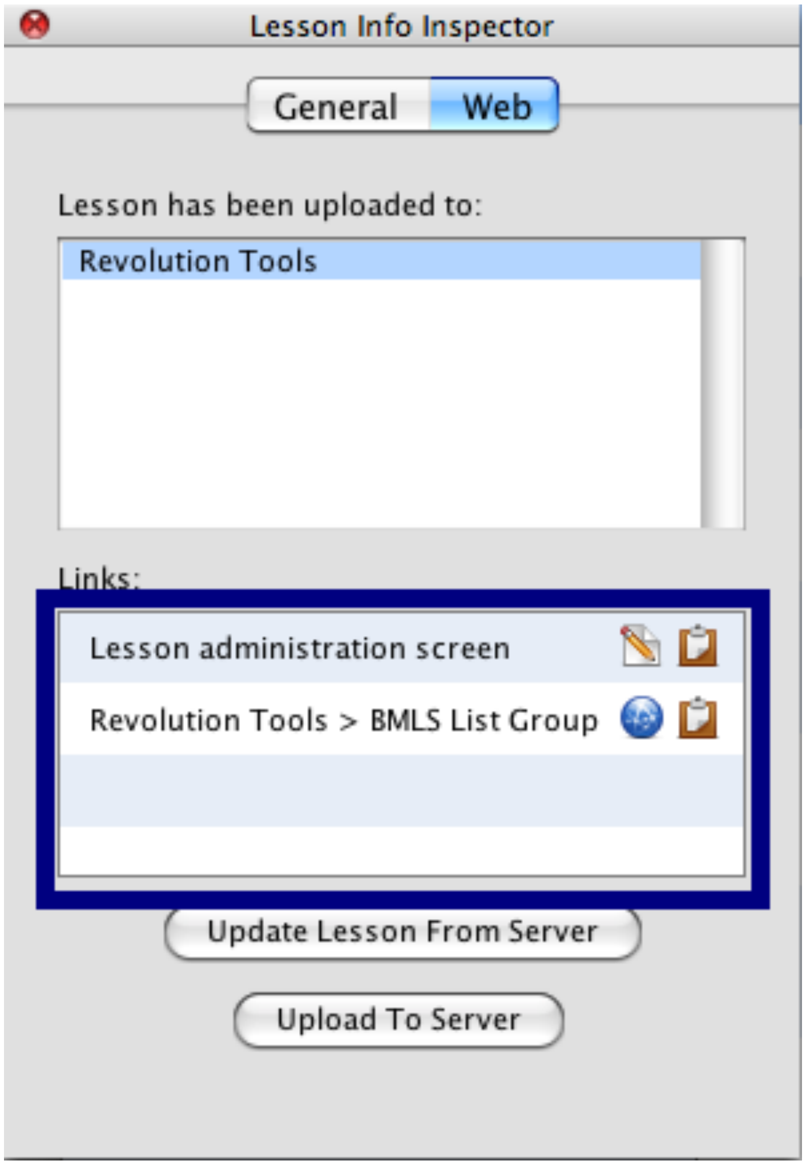


How do I Configure My ScreenSteps Software to Post to ScreenSteps Live?

Description

This lessons will show you how to configure your desktop software to post lessons to ScreenSteps Live. You must have signed up for an account at www.screenstepslive.com before you can do this.

## Data Grid Form



Lesson has been uploaded to:

Revolution Tools

Links:

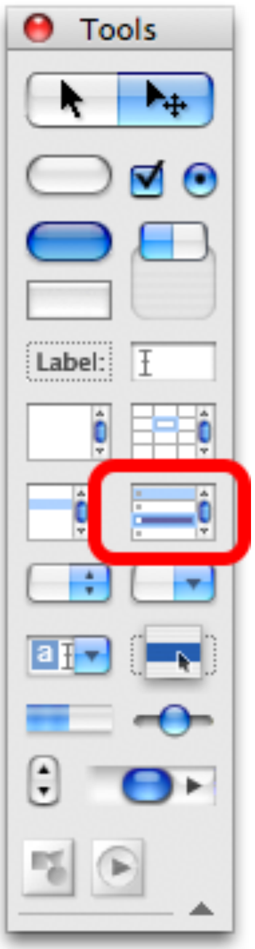
Lesson administration screen

Revolution Tools > BMLS List Group

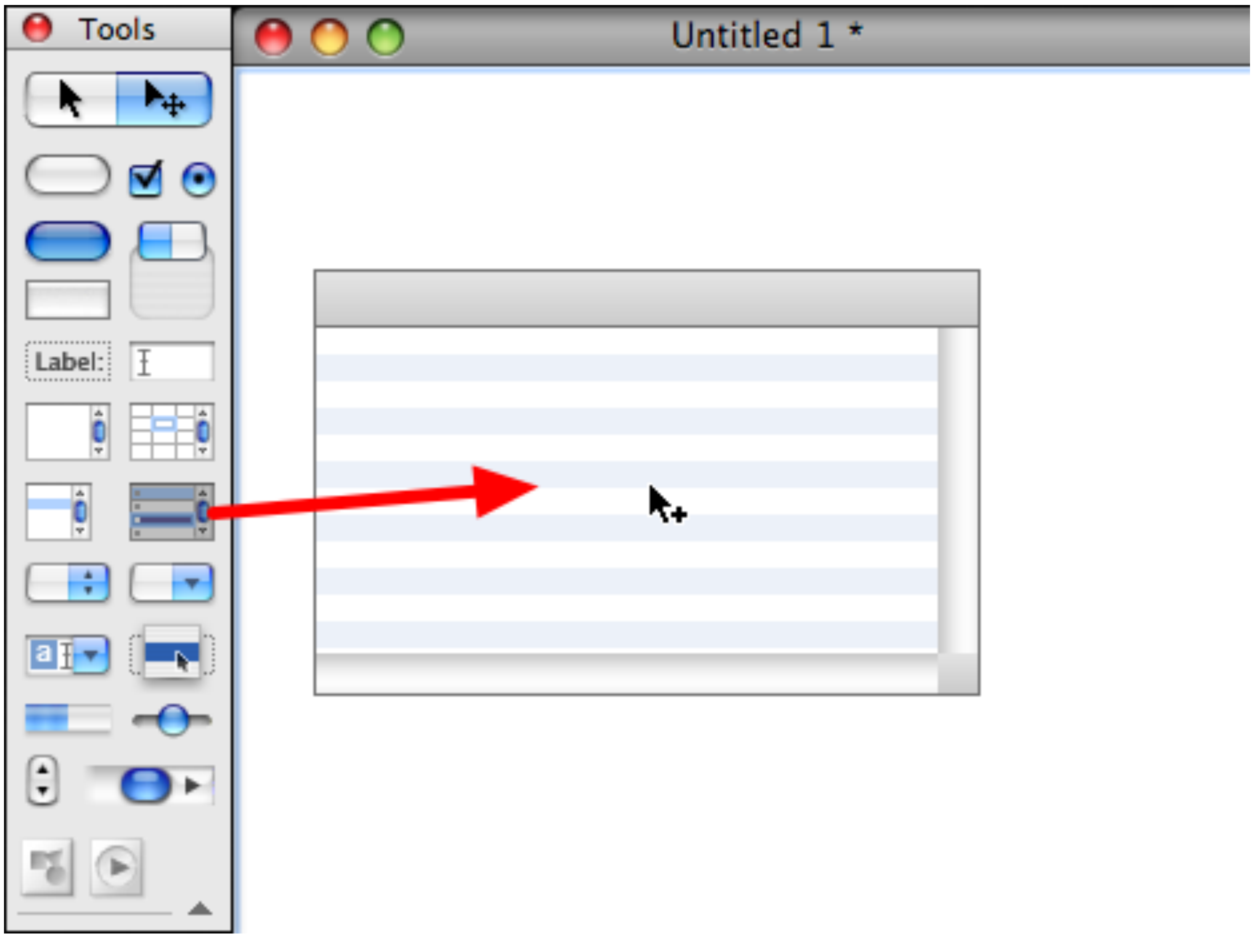
## How Do I Create My First Data Grid Table?

This lesson will show you how to create a bare bones data grid table. Data grid tables are useful when you need to display rows of data in structured columns.

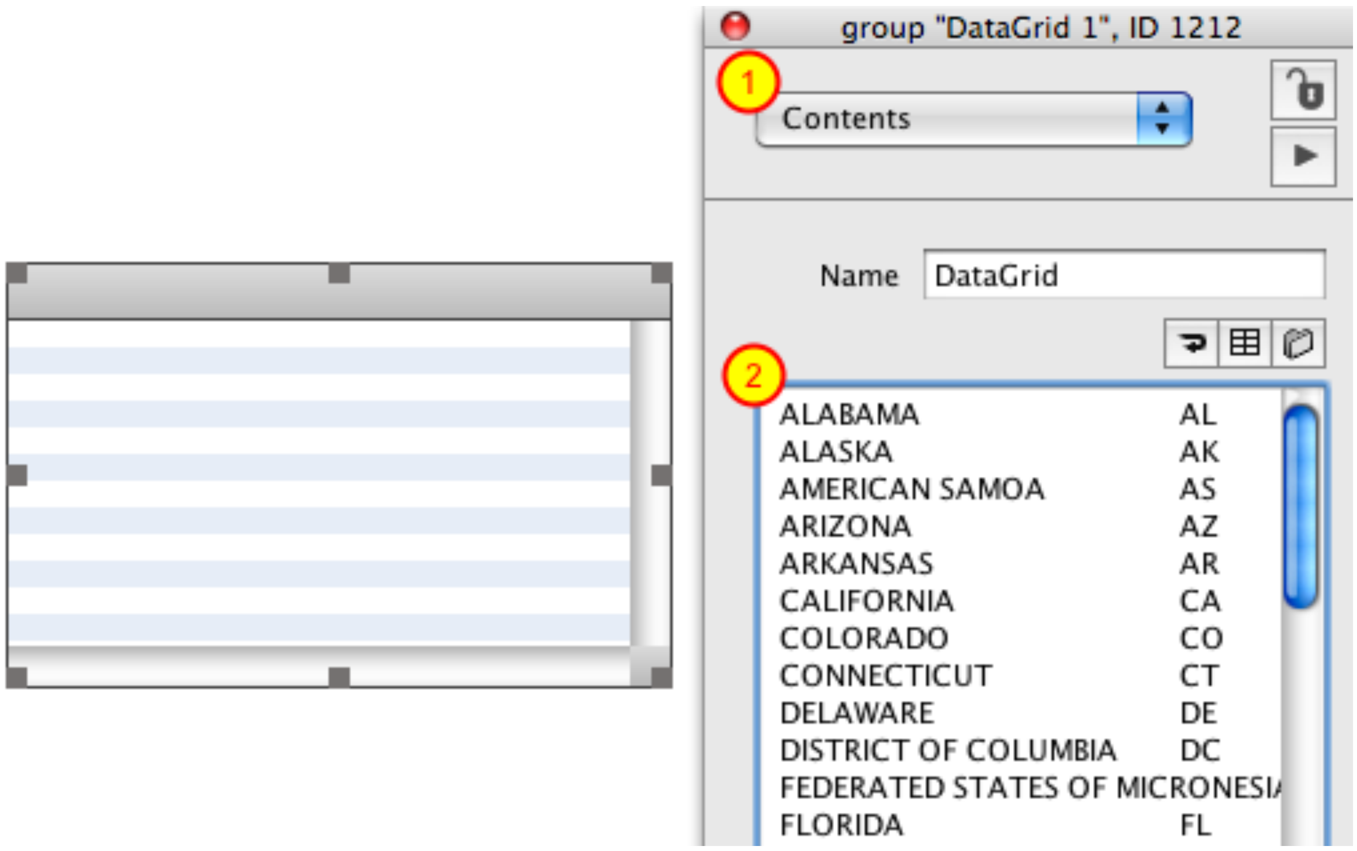
### Locate Data Grid on Tools Palette



## Drag Data Grid Onto Card



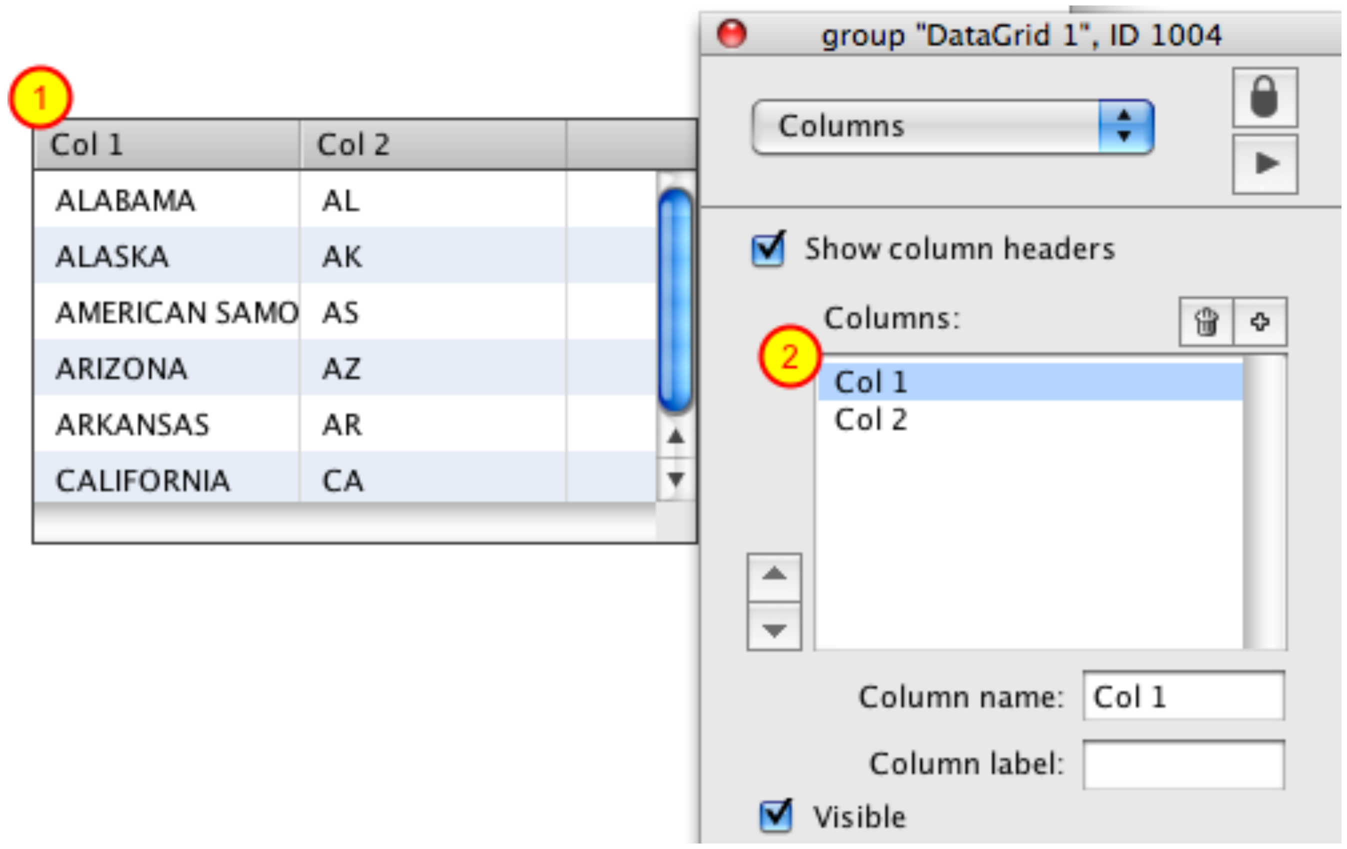
## Populate Data Grid Using Object Inspector



| | |
|---|---|
| ALABAMA | AL |
| ALASKA | AK |
| AMERICAN SAMOA | AS |
| ARIZONA | AZ |
| ARKANSAS | AR |
| CALIFORNIA | CA |
| COLORADO | CO |
| CONNECTICUT | CT |
| DELAWARE | DE |
| DISTRICT OF COLUMBIA | DC |
| FEDERATED STATES OF MICRONESI | |
| FLORIDA | FL |

Data Grid tables display data in columns that you can customize the look of. You don't have to worry about customization right now though because a Data Grid table can display plain text just fine. Let's test it out.

1) Switch to the **Contents** pane of the Object Inspector.
2) Type or paste some tab delimited text into the field. Click out of the field (in the Name field for example) to save the text you entered.



| Col 1 | Col 2 |
|---|---|
| ALABAMA | AL |
| ALASKA | AK |
| AMERICAN SAMO | AS |
| ARIZONA | AZ |
| ARKANSAS | AR |
| CALIFORNIA | CA |

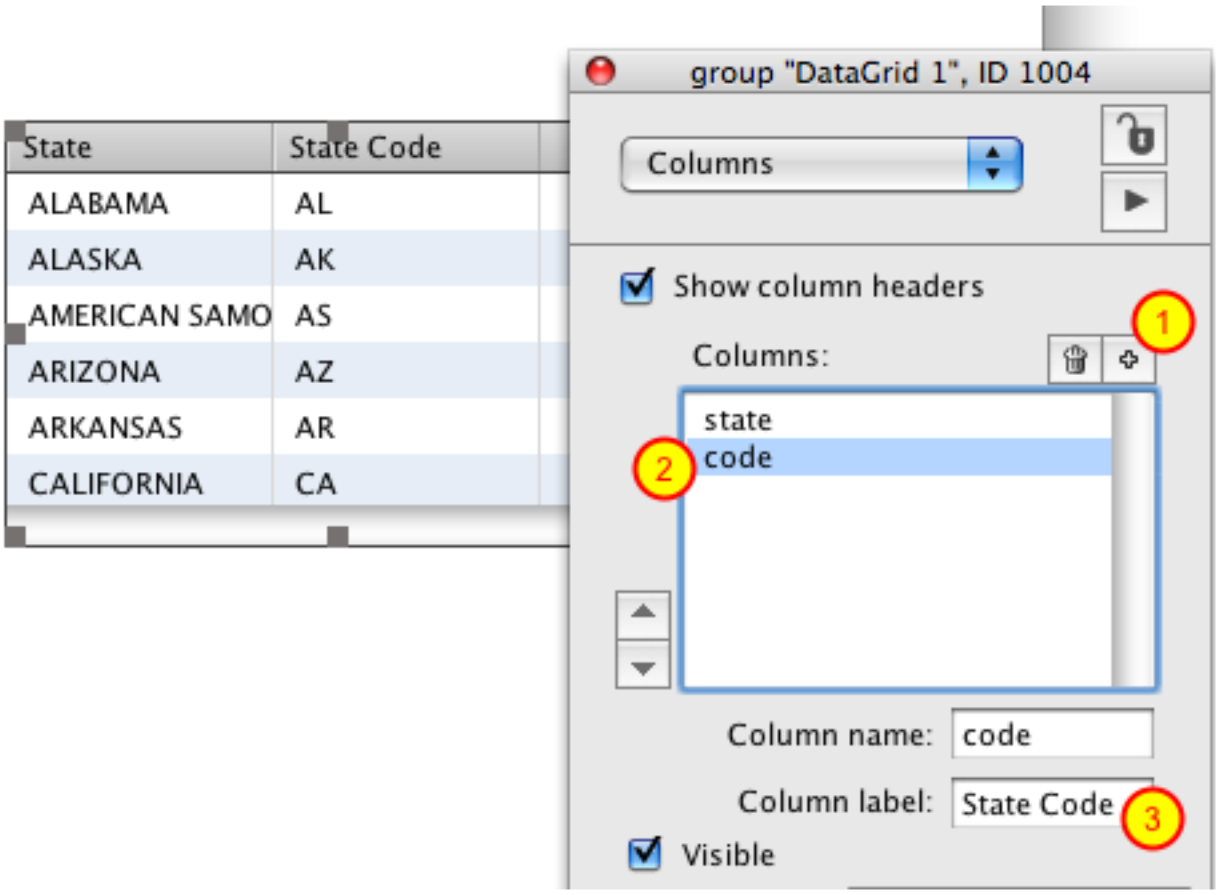
Columns: Col 1, Col 2

Column name: Col 1

Column label:

The tab delimited text you entered in the field will appear as columns in the data grid (1). The Object Inspector automatically creates a column in the Data Grid for each column in the text you provide (2).

## Customizing Columns



| State | State Code |
|---|---|
| ALABAMA | AL |
| ALASKA | AK |
| AMERICAN SAMO | AS |
| ARIZONA | AZ |
| ARKANSAS | AR |
| CALIFORNIA | CA |

Columns: state, code

Column name: code

Column label: State Code

Now that you've populate a Data Grid Table with some data let's look at the columns in the Data Grid.

Normally when you work with a Data Grid Table you will create columns in the property inspector in advance using the "+" button (1). Seeing as the columns I need already exist I've gone through and renamed "Col 1" to "state" and "Col 2" to "code" (2).

Note that I have assigned a label for the columns (3). Labels are useful for customizing the column labels in the Data Grid Table.

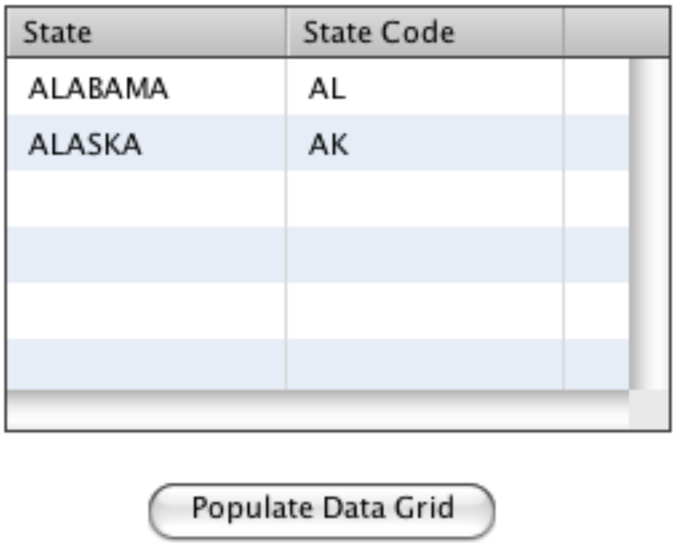
## Populate Data Grid Using dgText Property

Now that we have defined our columns let's look at how to populate a Data Grid Table by setting the dgText property. Here is an example handler with comments that you can place in a button.

**Note:** Verify that the name of your data grid matches the name used in the code below otherwise an error will occur when you try to run it.

```
on mouseUp
   ## Create tab delimited data.
   ## Note that first line has name of columns.
   ## Providing names tells Data Grid how to map
   ## data to appropriate columns.
   put "state" & tab & "code" & cr & \
          "ALABAMA" & tab & "AL" & cr & \
          "ALASKA" & tab & "AK" into theText

   ## Let Data Grid know that first line has column names
   put true into firstLineContainsColumnNames
   set the dgText [ firstLineContainsColumnNames ] of group "DataGrid" to theText
end mouseUp
```

Executing the above code would give you the following result in the Data Grid table.



| State | State Code |
|---|---|
| ALABAMA | AL |
| ALASKA | AK |

## Populating the Data Grid Using dgData Property

Here is an example of populating the Data Grid Table by setting the dgData property to an array. The end result is the same as the previous step.

**Note:** Verify that the name of your data grid matches the name used in the code below otherwise an error will occur when you try to run it.

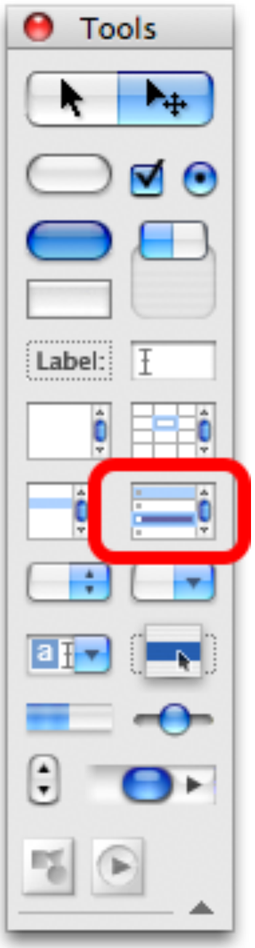
```
on mouseUp
   ## Create a nested array.
   ## Array key names in the 2nd dimension
   ## dictate values for individual columns
   put "ALABAMA" into theDataA[ 1 ][ "state" ]
   put "AL" into theDataA[ 1 ][ "code" ]
```

```
    put "ALASKA" into theDataA[ 2 ][ "state" ]
    put "AK" into theDataA[ 2 ][ "code" ]

    set the dgData of group "DataGrid" to theDataA
end mouseUp
```

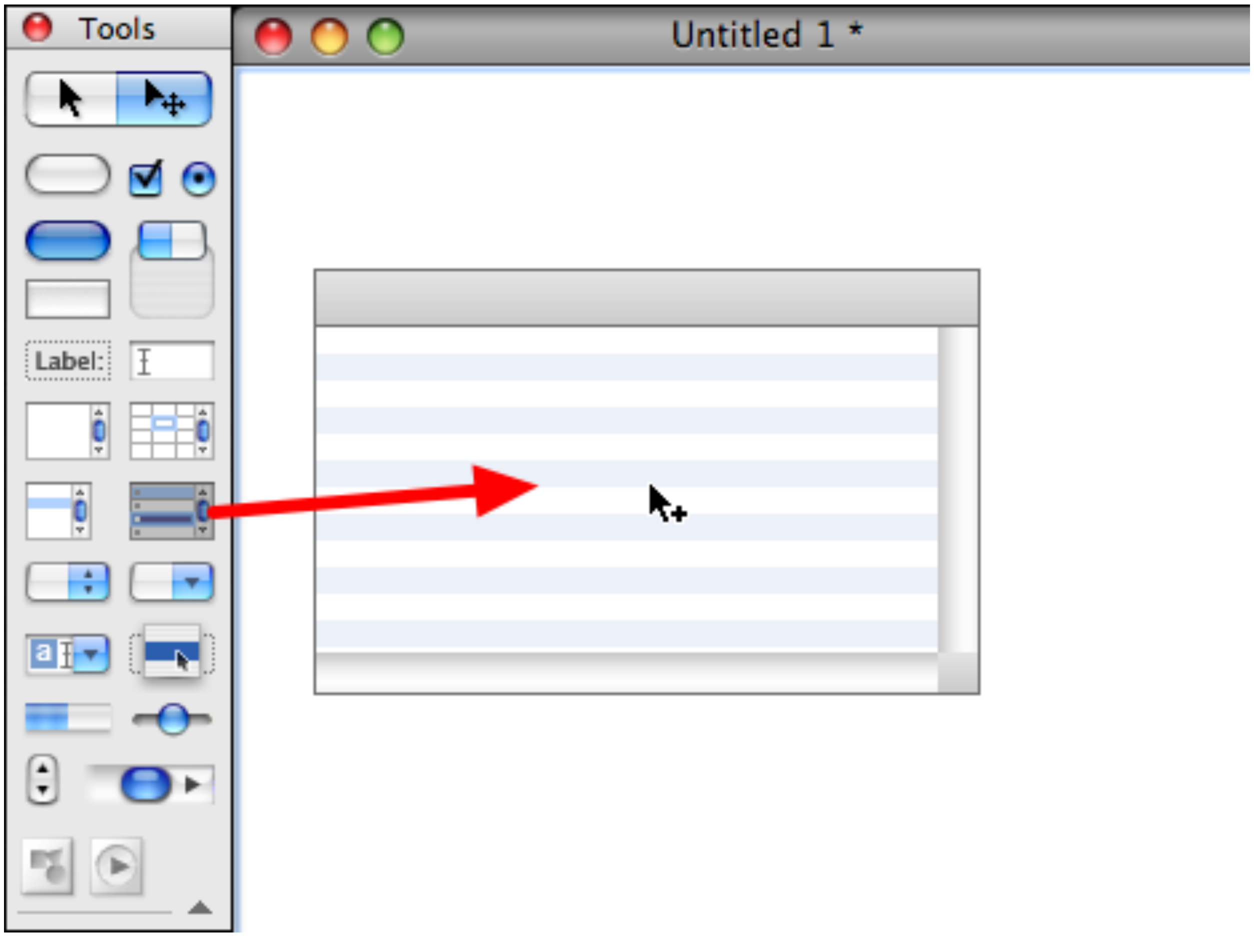
## How Do I Create My First Data Grid Form?

This lesson will show you how to create a bare bones data grid form. Data grid forms are useful when you need a less rigid layout than columns in a table provide. A data grid form gives you complete control over the look and feel of each record you display.

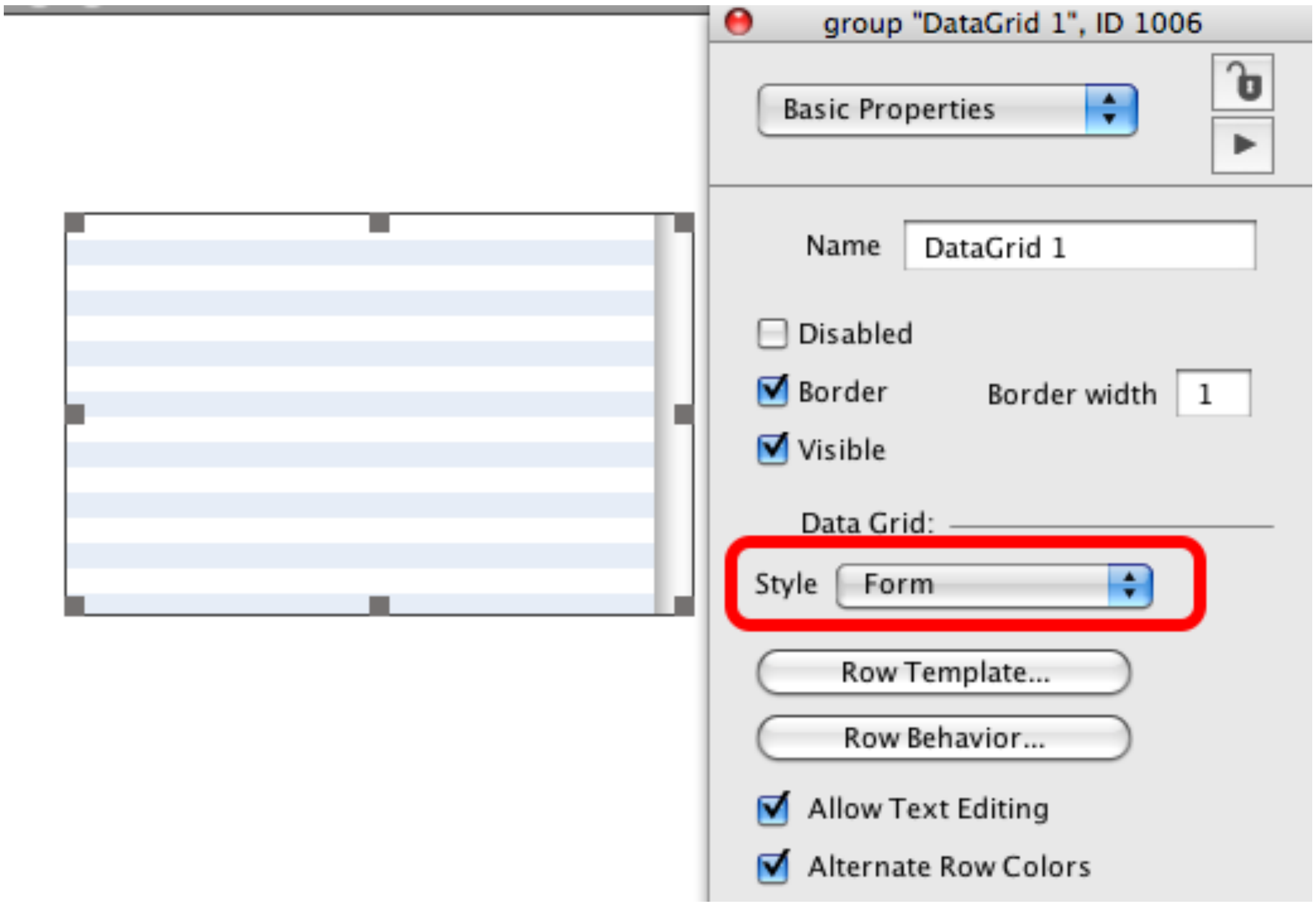
### Locate Data Grid on Tools Palette



## Drag Data Grid Onto Card



## Change Style to "form"



group "DataGrid 1", ID 1006

Basic Properties

Name: DataGrid 1

Disabled [ ]

Border [✓] Border width 1

Visible [✓]

Data Grid:

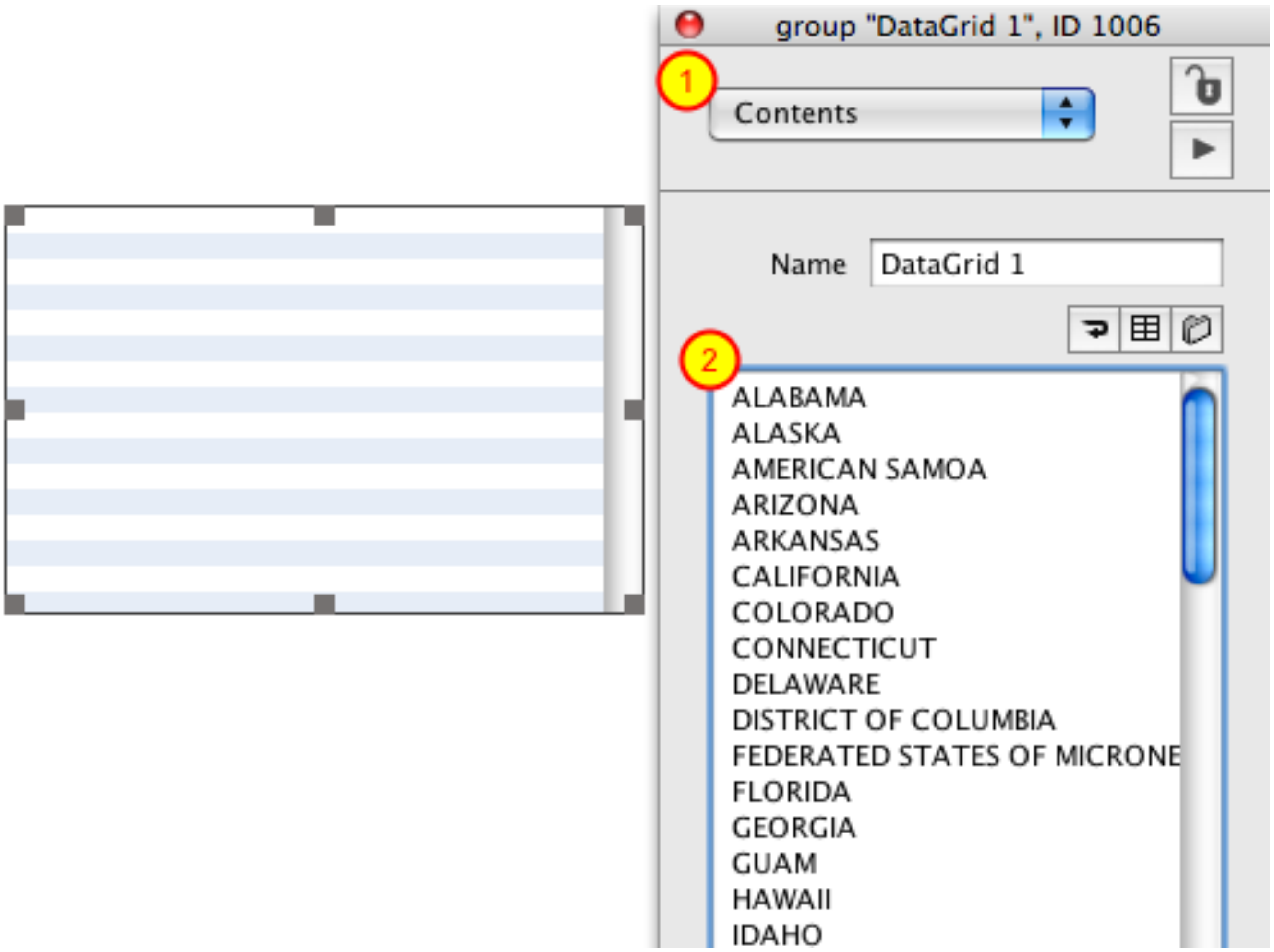
Style: Form

Allow Text Editing [✓]

Alternate Row Colors [✓]

Select the Data Grid and open the Revolution property inspector. Change the **style** to *form*.

## Add Some Text

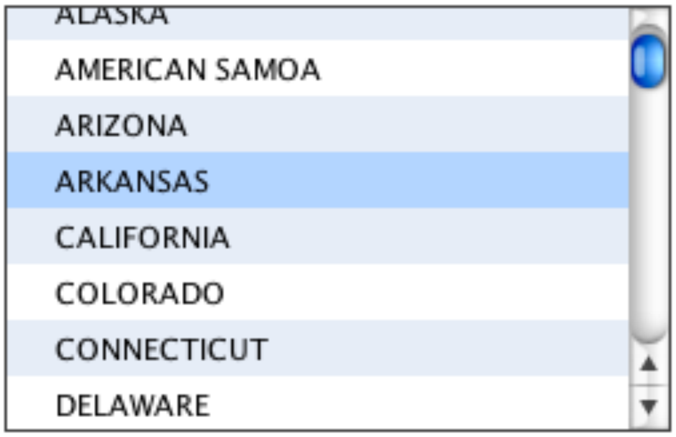


```
ALABAMA
ALASKA
AMERICAN SAMOA
ARIZONA
ARKANSAS
CALIFORNIA
COLORADO
CONNECTICUT
DELAWARE
DISTRICT OF COLUMBIA
FEDERATED STATES OF MICRONE
FLORIDA
GEORGIA
GUAM
HAWAII
IDAHO
```

Now you can assign some text to the data grid. By default a data grid will render some basic data without you having to customize the **row template**. To assign some text:

1) Switch to the **Contents** pane of the property inspector.
2) Type or paste some line delimited text into the field.

## Result



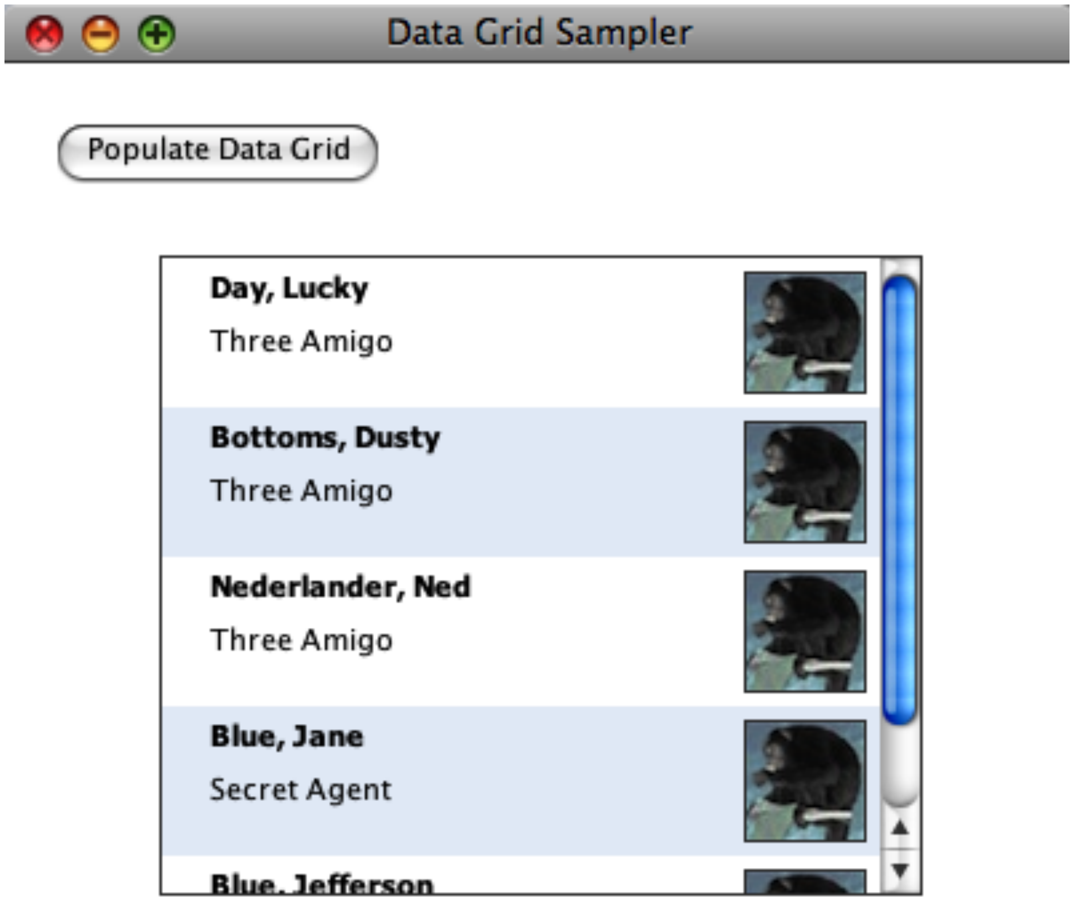
That's it! The data you entered in the field will appear in the data grid. Now you are ready to customize the look and feel of your records by modifying the **row template**.

You should note that while you can assign line delimited text to a data grid form, the data grid really works with multi-dimensional arrays under the hood. By using arrays you can store and display all different kinds of data in a data grid. Being able to assign the uText property is merely a convenience.

## Example: Creating a List of People

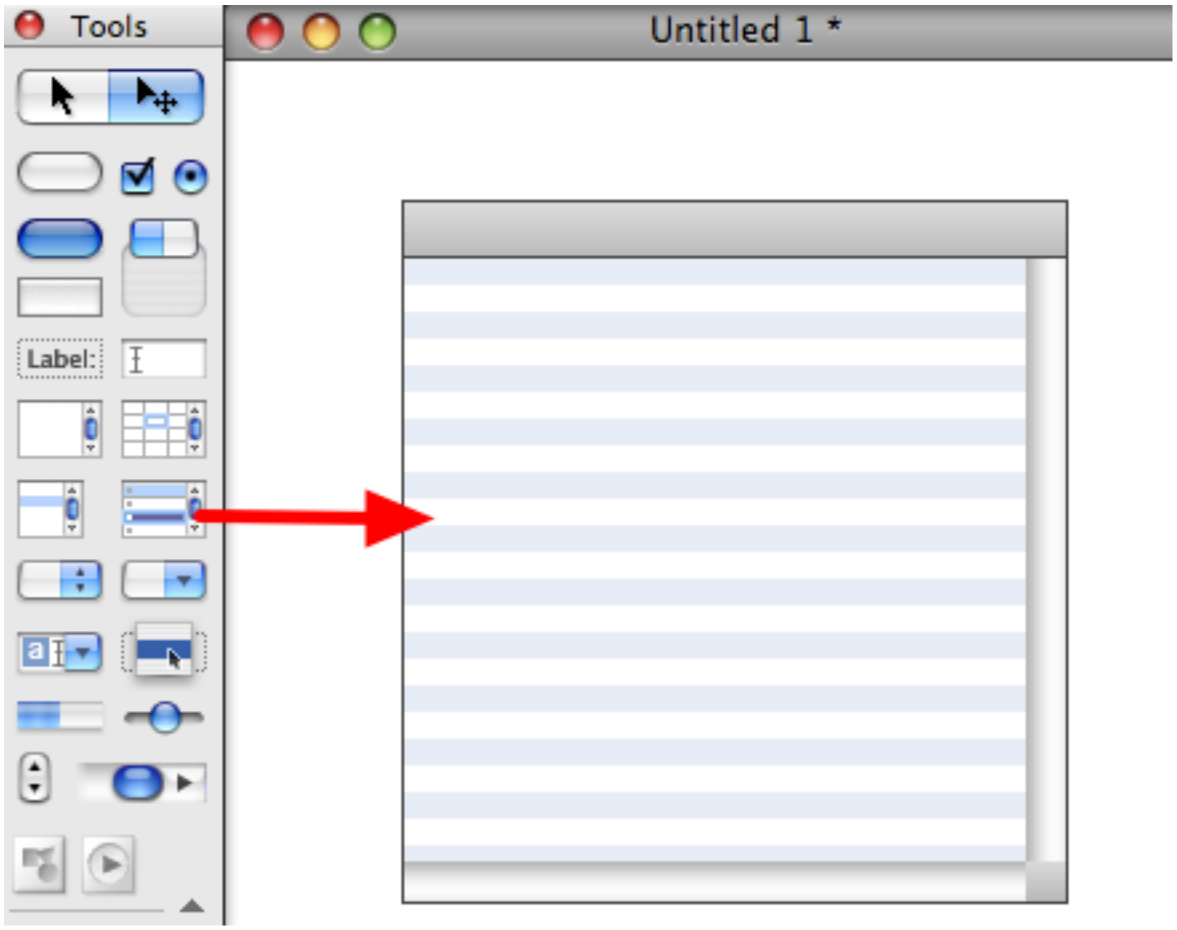
This lesson provides a low detail, step-by-step example of creating a custom data grid form. You can download the attached sample stack to see the scripts (or download it here).

### What You Will Create



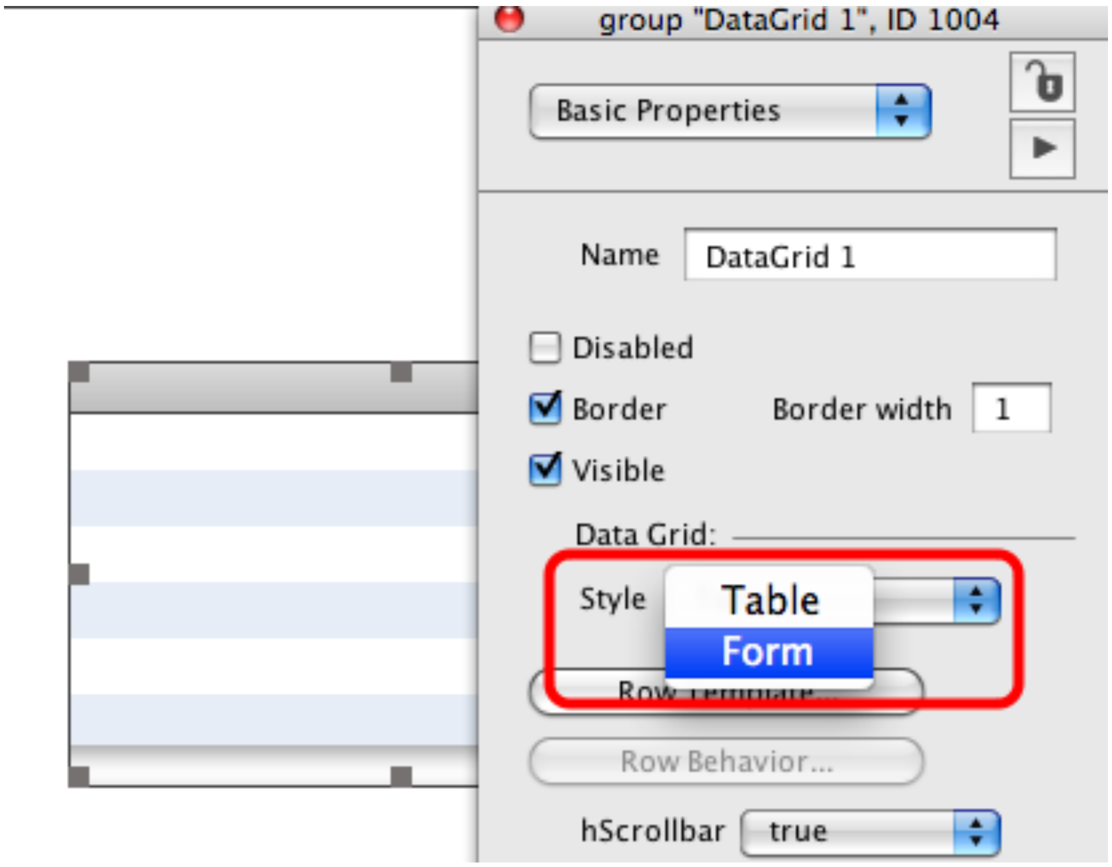
This is the data grid that you will create. Each row displays a name, a title and an image.

## Add Data Grid to Card



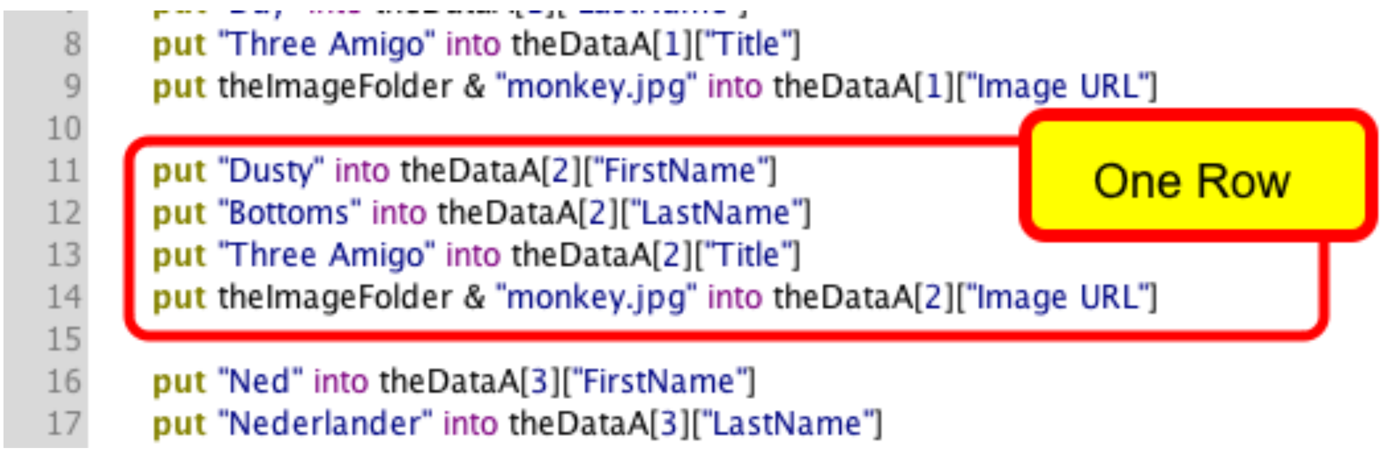
To begin, drag a data grid from the Tools palette onto  your card.

## Change Data Grid Style


group "DataGrid 1", ID 1004

Select the data grid and from the **Basic Properties** pane of the property inspector change the *style* of the data grid to **Form**.

## The Data That Is Going to Be Displayed In Each Row



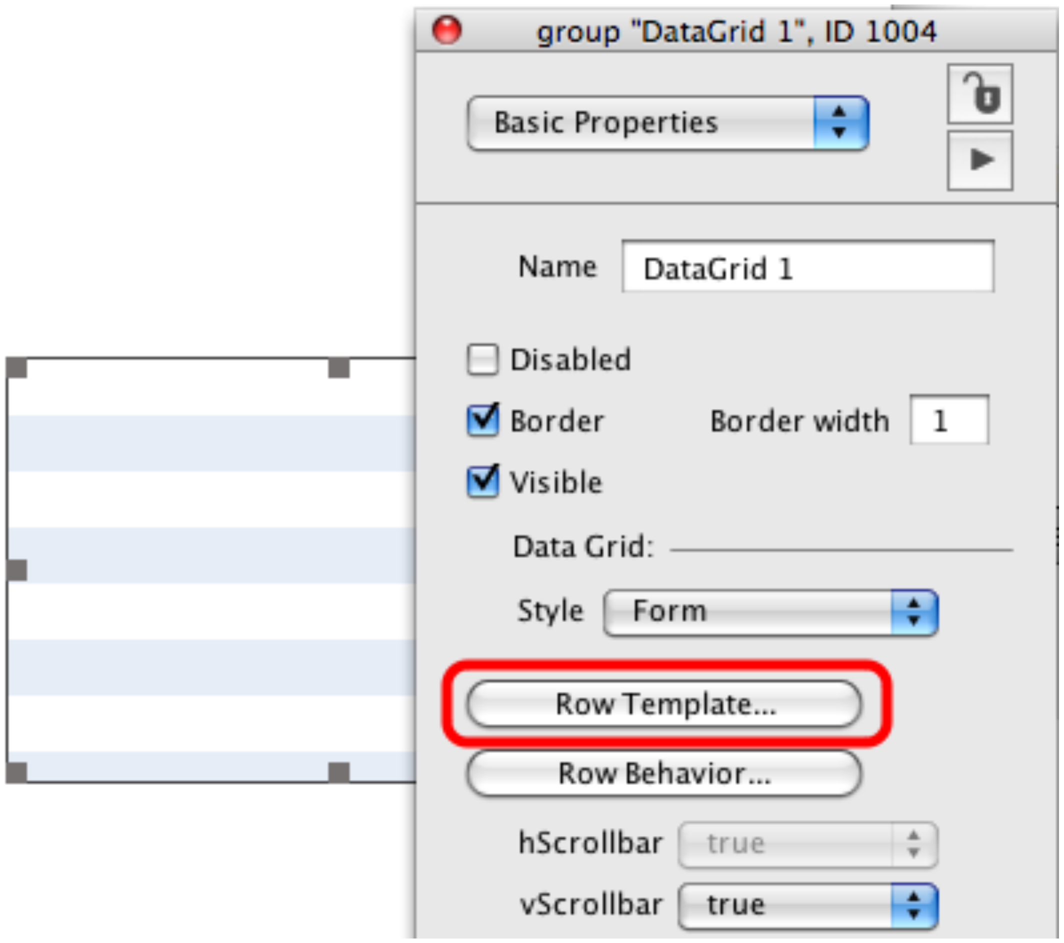
```
put "Three Amigo" into theDataA[1]["Title"]
put theImageFolder & "monkey.jpg" into theDataA[1]["Image URL"]

put "Dusty" into theDataA[2]["FirstName"]
put "Bottoms" into theDataA[2]["LastName"]
put "Three Amigo" into theDataA[2]["Title"]
put theImageFolder & "monkey.jpg" into theDataA[2]["Image URL"]

put "Ned" into theDataA[3]["FirstName"]
put "Nederlander" into theDataA[3]["LastName"]
```

Now you are going to design the row template for the data grid form. The row template dictates how each record in the data grid will be displayed.

Before you start editing the row template you need to know about the data that will be displayed each row. This screenshot shows an excerpt from the code that populates the data grid using a multi-dimensional array (look in the card script of the example stack). Each row will have access to the data stored in the *FirstName, LastName, Title* and *ImageURL* keys of the array.

You are now going to add UI controls to the row template in order to display the data for each of these keys.

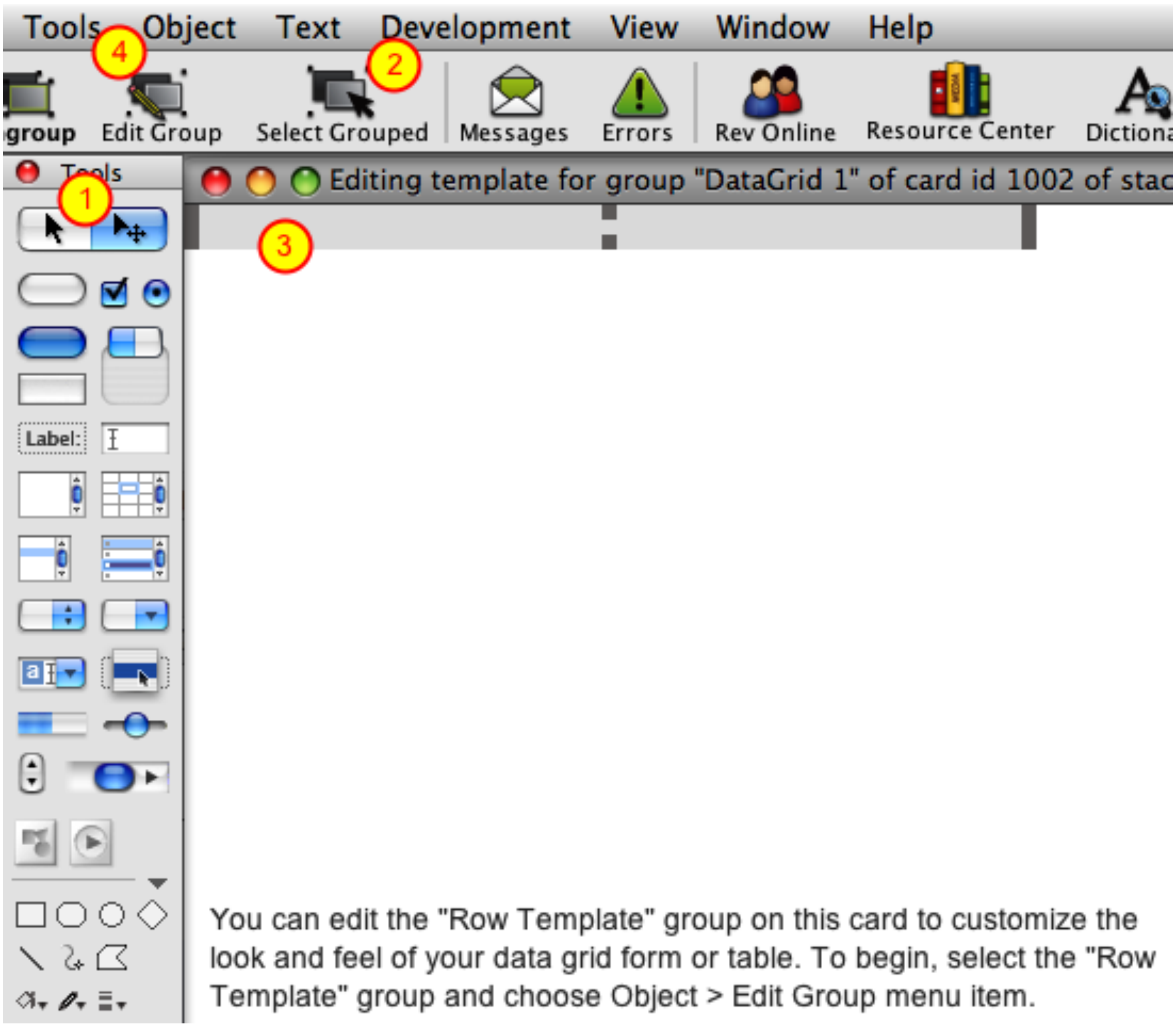
## Edit the Row Template



Name: DataGrid 1
Disabled: unchecked
Border: checked
Border width: 1
Visible: checked
Data Grid:
Style: Form
hScrollbar: true
vScrollbar: true

With the data grid selected, click on the **Row Template** button in the property inspector to open the card containing row template controls.

## Edit the Row Template Group



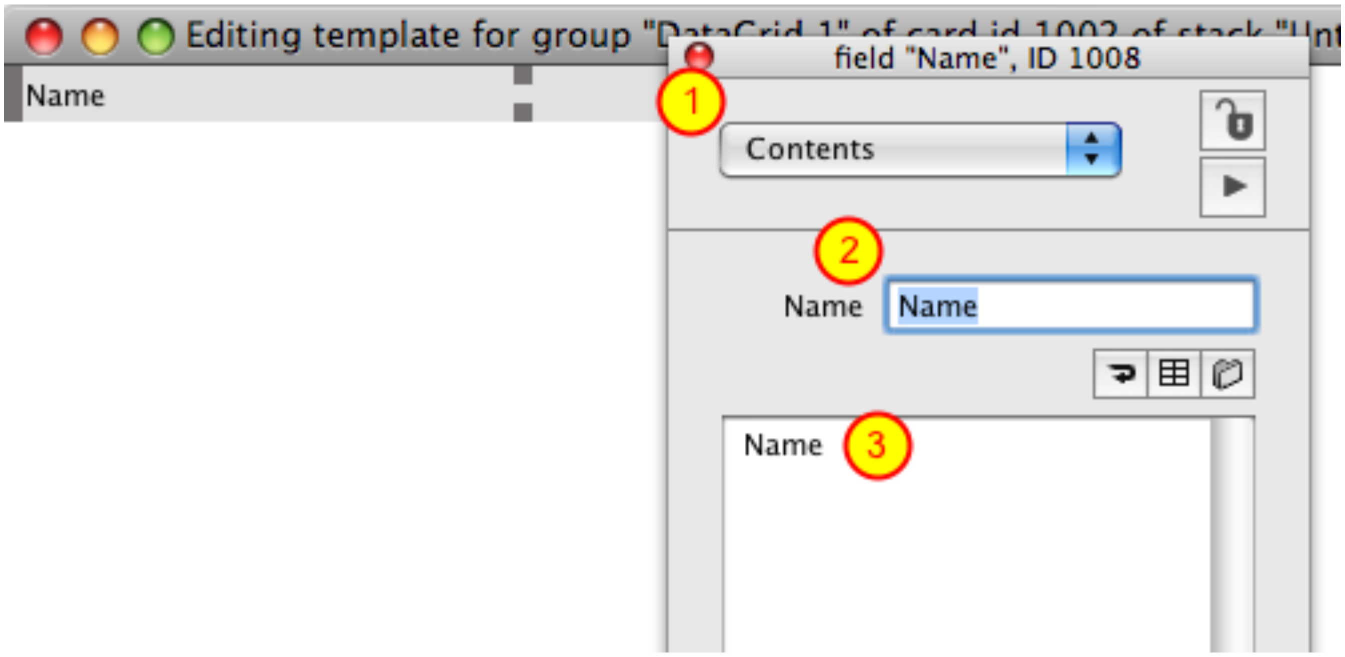
Editing template for group "DataGrid 1" of card id 1002 of stac

You can edit the "Row Template" group on this card to customize the look and feel of your data grid form or table. To begin, select the "Row Template" group and choose Object > Edit Group menu item.

In order to customize the look we need to edit the contents of the template group.

1) Make sure the **Edit** tool is selected.
2) Make sure that **Select Grouped** IS NOT active.
3) Select the row template group.
4) Click **Edit Group** (4) in the Revolution toolbar.

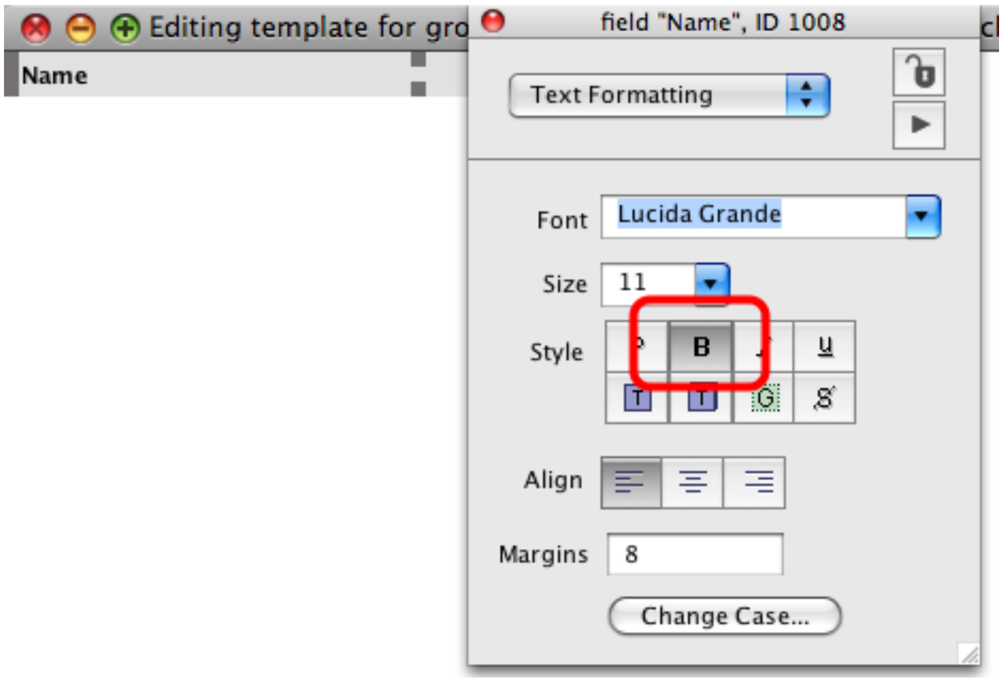
## Rename Label Field


Editing template for group "DataGrid 1" of card id 1002 of stack "Unt
Name
field "Name", ID 1008
1
Contents
2
Name
Name
Name
3

The default template group has a field named "Label". Click on the left side of the gray rectangle to select it.

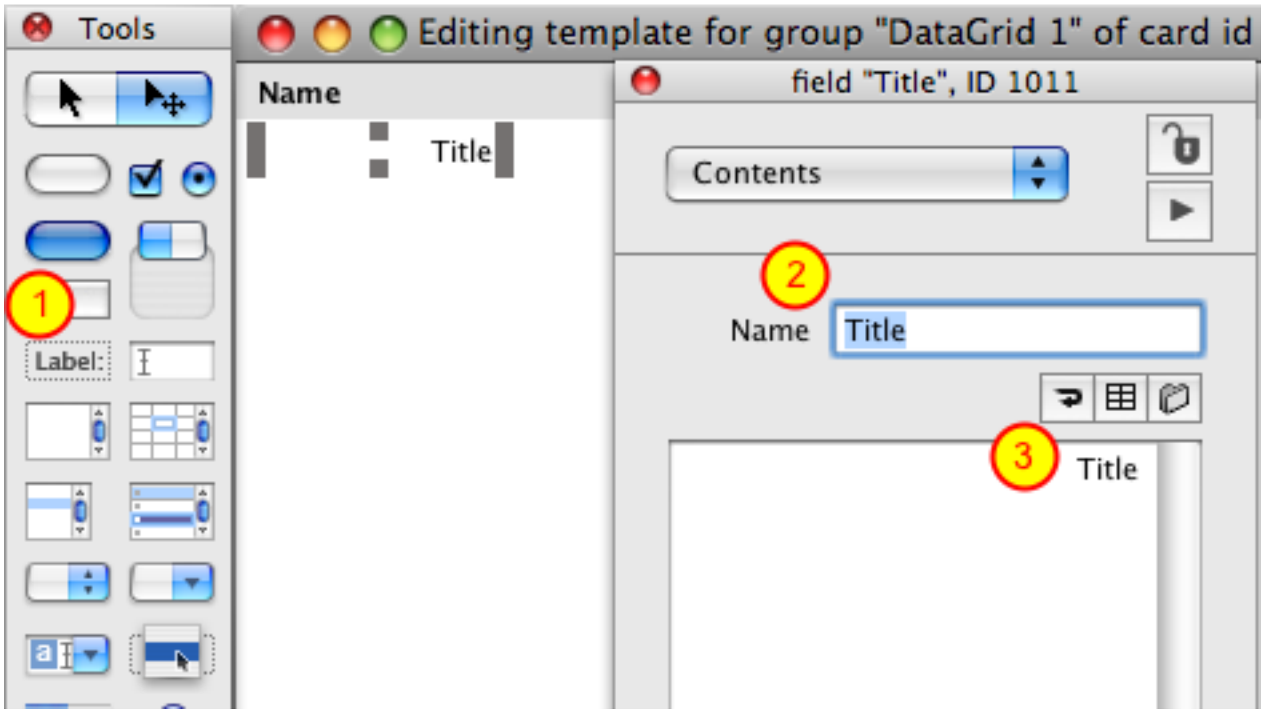
Open the property inspector and from the **Contents** pane (1) change the name of the field to **Name (2)** and assign **Name** as the content (3).

## Make Name Field Bold



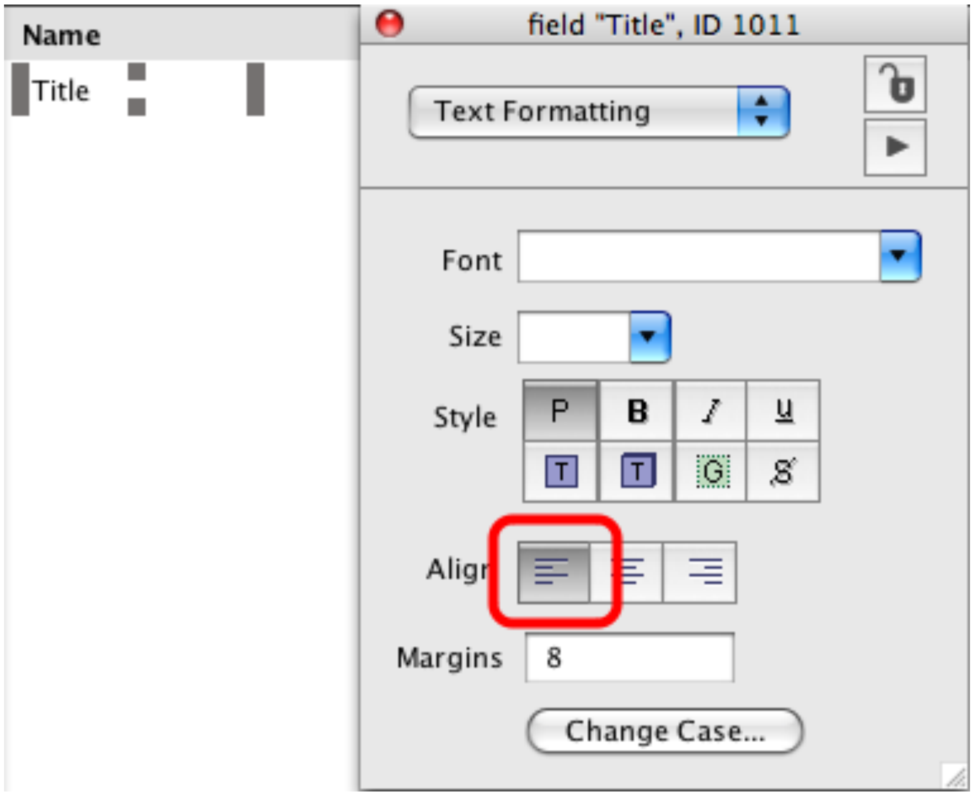
With the **Name** field still selected change the **Style** to **Bold** from the **Text Formatting** pane of the property inspector.

## Add Title Field



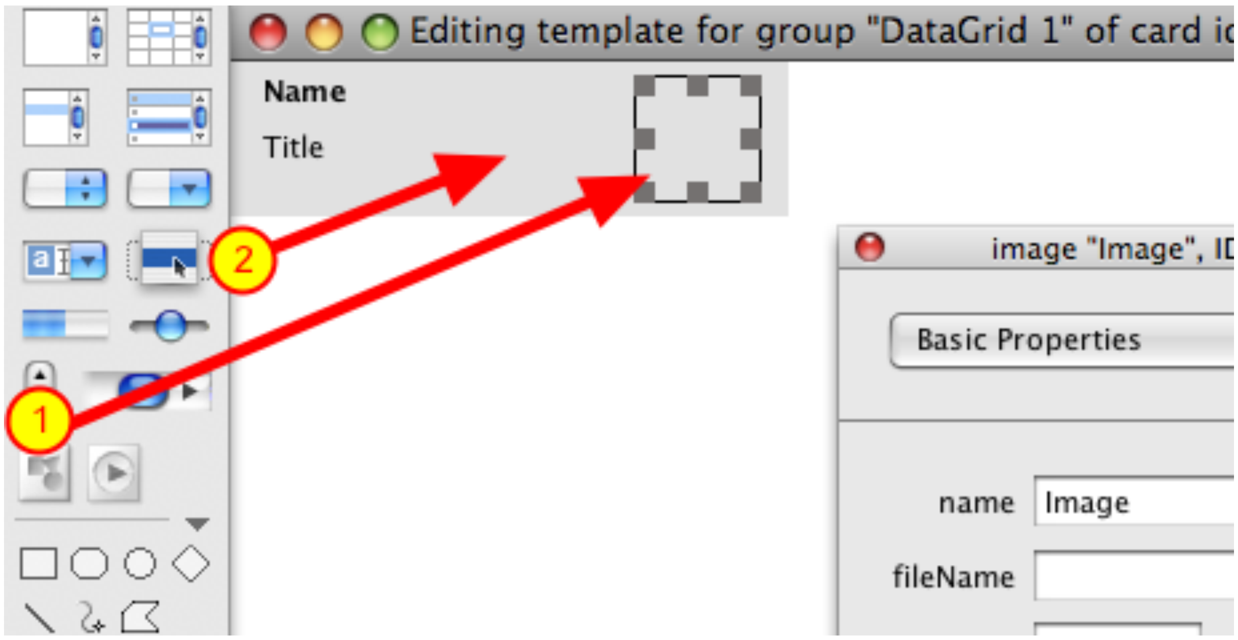
Drag a label field from the Tools palette (1). Name it **Title** (2) and assign **Title** to the content (3).

## Change Text Alignment



Change the text alignment of the field to the left side using the **Text Formatting** pane of the property inspector.

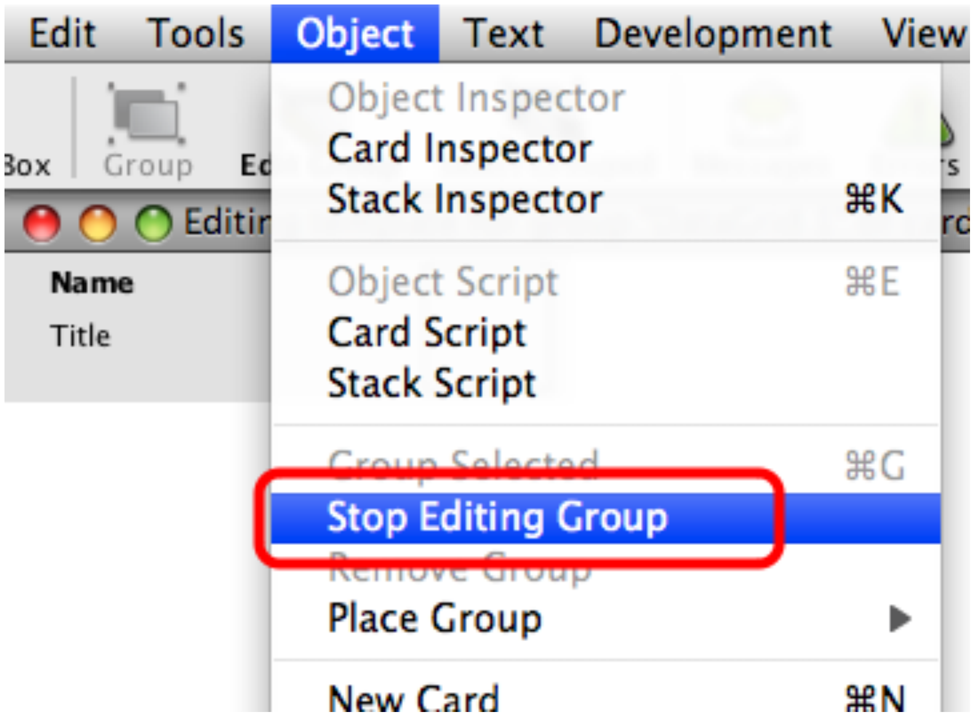
## Add An Image and Resize The Background Graphic



Now drag an image object into the group using the Tools palette (1). With the image selected, set the width and height to 48 pixels using the **Size & Position** pane in the property inspector.

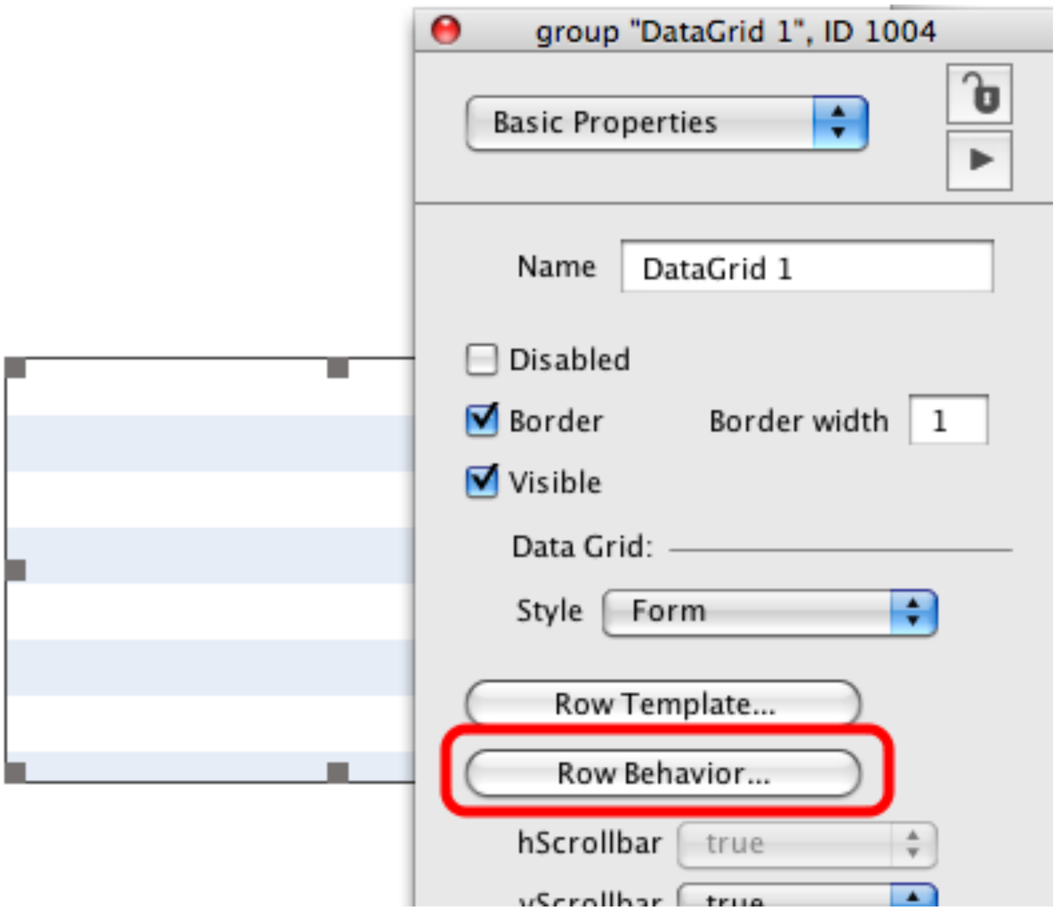
Resize the "Background" graphic so that it frames the other controls (2). In the example stack the width of the graphic is 214 and the height is 56.

## Stop Editing the Row Template Group



You are now done adding the UI objects to the row template group. From the **Object** menu, select **Stop Editing Group**.

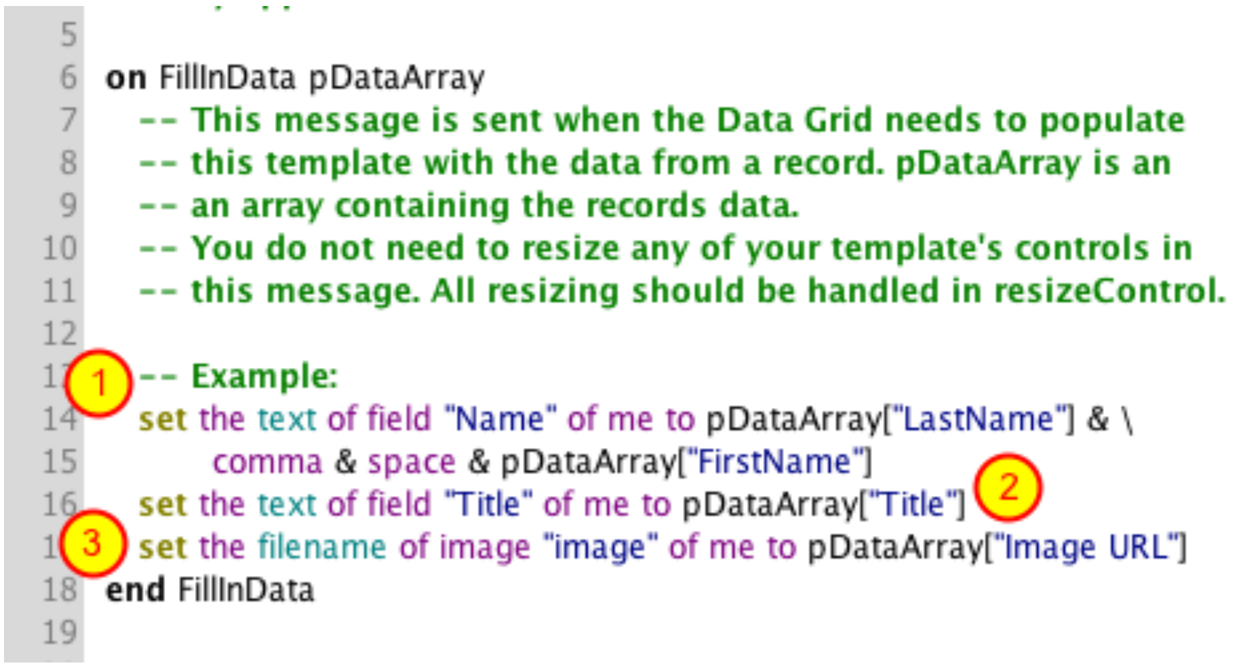
## Edit the Row Behavior



Now that you have designed the controls for each row you need to write the script that will move data into the controls and position them on the screen.

Click on the **Row Behavior** button in the property inspector to begin editing the row behavior script.

## The FillInData Message



```
on FillInData pDataArray
  -- This message is sent when the Data Grid needs to populate
  -- this template with the data from a record. pDataArray is an
  -- an array containing the records data.
  -- You do not need to resize any of your template's controls in
  -- this message. All resizing should be handled in resizeControl.

  -- Example:
  set the text of field "Name" of me to pDataArray["LastName"] & \
        comma & space & pDataArray["FirstName"]
  set the text of field "Title" of me to pDataArray["Title"]
  set the filename of image "image" of me to pDataArray["Image URL"]
end FillInData
```

In the script that opens when you click the Row Behavior button you will find a command named **FillInData**. This is where you move data from the pDataArray parameter into the controls you created.

pDataArray is an array with the keys *FirstName, LastName, Name* and *Title*. The code that moves the values from the array into the UI controls is pretty straightforward.

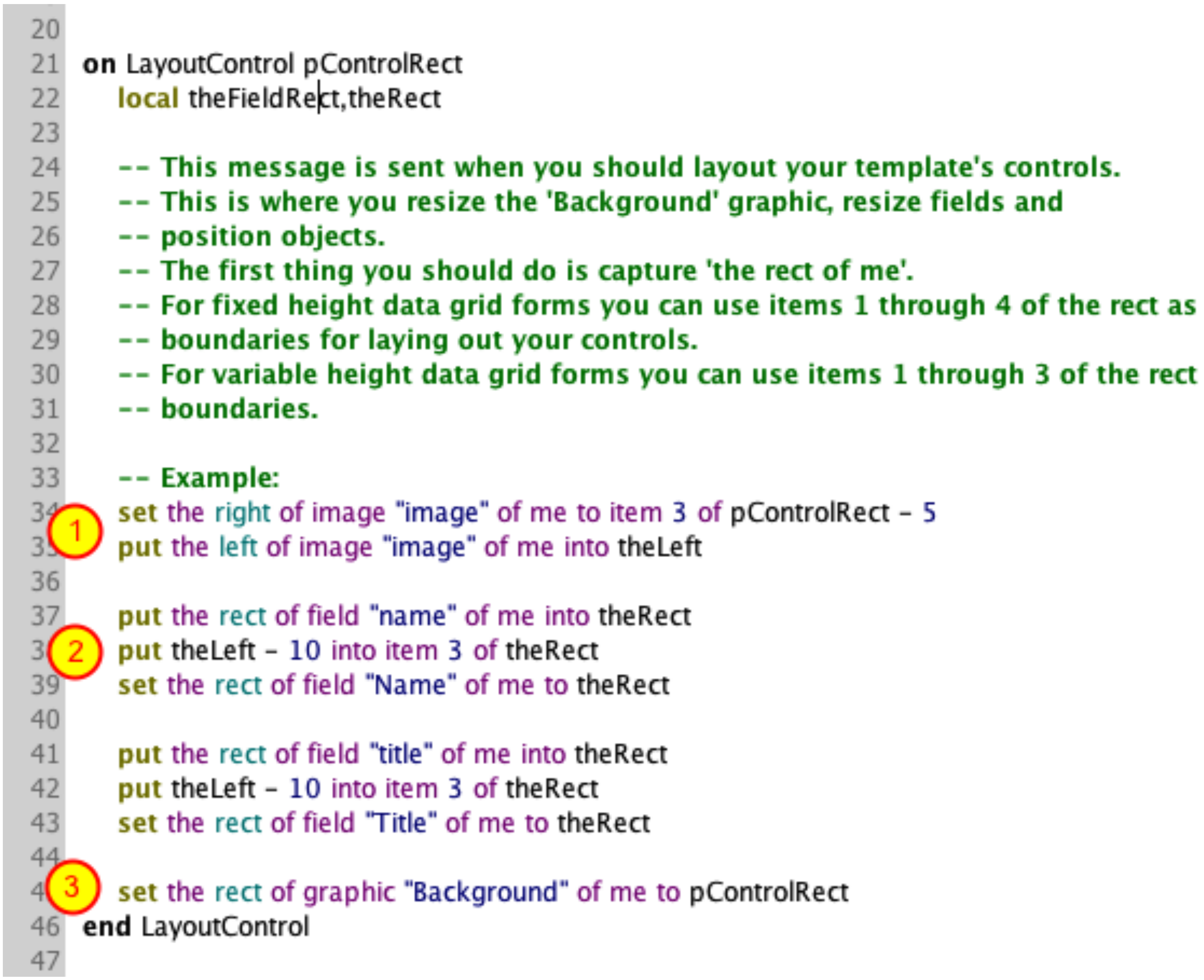
1) Assign the values of the FirstName and LastName keys to the **Name** field.
2) Assign the value of the Title key to the **Title** field.
3) Assign the value of the Image URL key to the filename property of the image control.

You don't need to worry about positioning any of your controls in the FillInData command. You just need to write the code that assigns the data to the UI controls.

**==========**
**Copy & Paste**
**==========**

```
on FillInData pDataArray
   -- This message is sent when the Data Grid needs to populate
   -- this template with the data from a record. pDataArray is an
   -- an array containing the records data.
   -- You do not need to resize any of your template's controls in
   -- this message. All resizing should be handled in resizeControl.

   -- Example:
   set the text of field "Name" of me to pDataArray["LastName"] & \
       comma & space & pDataArray["FirstName"]
   set the text of field "Title" of me to pDataArray["Title"]
   set the filename of image "image" of me to pDataArray["Image URL"]
end FillInData
```

## The LayoutControl Message



```

on LayoutControl pControlRect
   local theFieldRect,theRect

   -- This message is sent when you should layout your template's controls.
   -- This is where you resize the 'Background' graphic, resize fields and
   -- position objects.
   -- The first thing you should do is capture 'the rect of me'.
   -- For fixed height data grid forms you can use items 1 through 4 of the rect as
   -- boundaries for laying out your controls.
   -- For variable height data grid forms you can use items 1 through 3 of the rect
   -- boundaries.

   -- Example:
   set the right of image "image" of me to item 3 of pControlRect - 5
   put the left of image "image" of me into theLeft

   put the rect of field "name" of me into theRect
   put theLeft - 10 into item 3 of theRect
   set the rect of field "Name" of me to theRect

   put the rect of field "title" of me into theRect
   put theLeft - 10 into item 3 of theRect
   set the rect of field "Title" of me to theRect

   set the rect of graphic "Background" of me to pControlRect
end LayoutControl

```

The **LayoutControl** message is where you position all of your controls. For this template you begin by positioning the image (1). Next you extend the "Name" and "Title" fields to the edge of the image (2). Finally you resize the "Background" graphic to fill the entire row rectangle.

**==========**
**Copy & Paste**
**==========**

```
on LayoutControl pControlRect
```

```
    local theFieldRect,theRect

    -- This message is sent when you should layout your template's controls.
    -- This is where you resize the 'Background' graphic, resize fields and
    -- position objects.
    -- The first thing you should do is capture 'the rect of me'.
    -- For fixed height data grid forms you can use items 1 through 4 of the rect as
    -- boundaries for laying out your controls.
    -- For variable height data grid forms you can use items 1 through 3 of the rect as
    -- boundaries.

    -- Example:
    set the right of image "image" of me to item 3 of pControlRect - 5
    put the left of image "image" of me into theLeft

    put the rect of field "name" of me into theRect
    put theLeft - 10 into item 3 of theRect
    set the rect of field "Name" of me to theRect

    put the rect of field "title" of me into theRect
    put theLeft - 10 into item 3 of theRect
    set the rect of field "Title" of me to theRect

    set the rect of graphic "Background" of me to pControlRect
end LayoutControl
```

## Set the Data of the Control

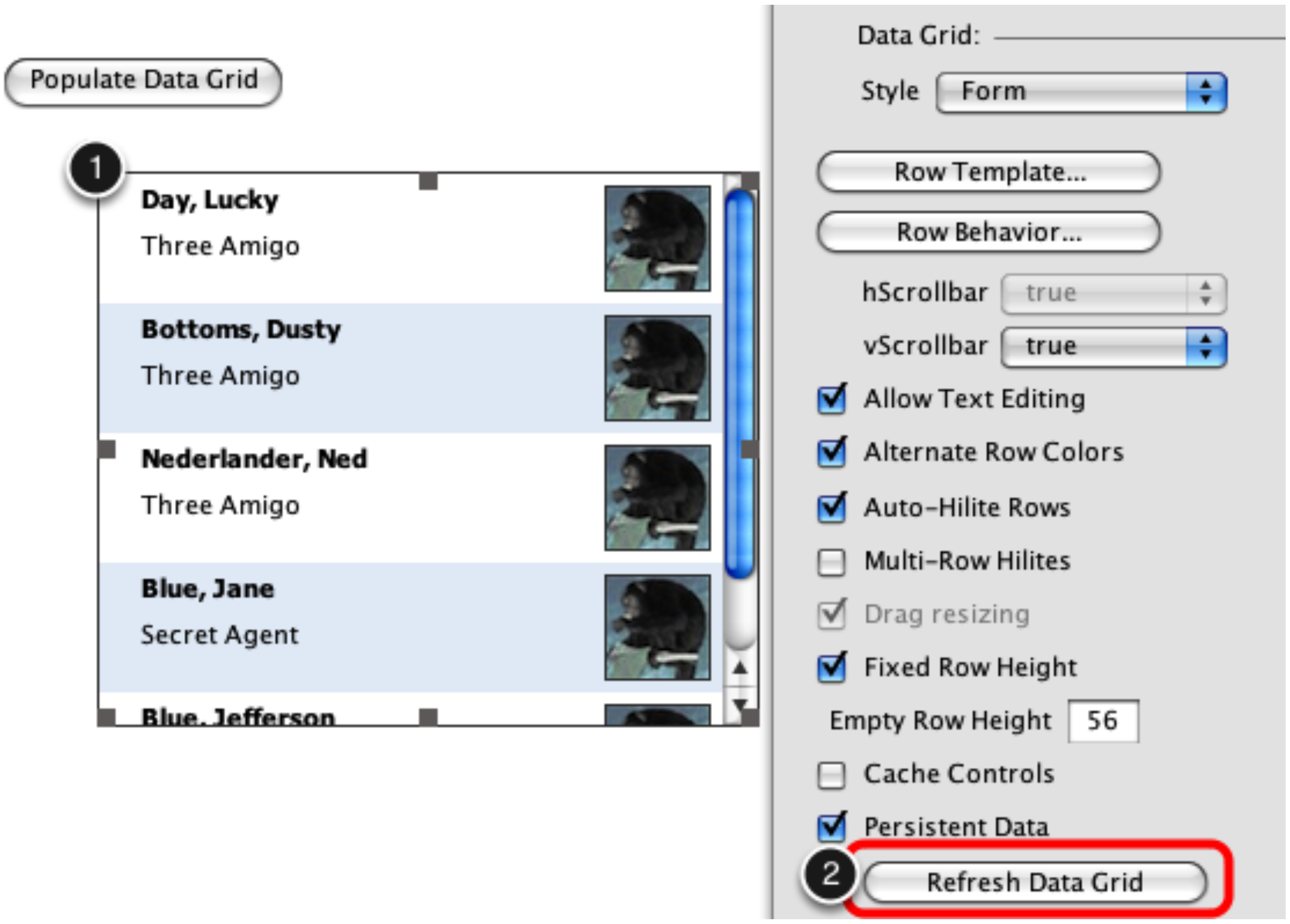
This is an example of a handler that populates the data grid with data by setting the dgData property. The dgData property accepts a multi-dimensional array. The first dimensions is an integer representing the row. The second dimension are the key/values you want to pass to each row.

You can find this handler in the card script of the example stack.

**==========**
**Copy & Paste**
**==========**

```
command uiPopulatePeople
    put "images/" into theImageFolder

    put "Lucky" into theDataA[1]["FirstName"]
    put "Day" into theDataA[1]["LastName"]
    put "Three Amigo" into theDataA[1]["Title"]
    put theImageFolder & "monkey.jpg" into theDataA[1]["Image URL"]

    put "Dusty" into theDataA[2]["FirstName"]
    put "Bottoms" into theDataA[2]["LastName"]
    put "Three Amigo" into theDataA[2]["Title"]
    put theImageFolder & "monkey.jpg" into theDataA[2]["Image URL"]

    put "Ned" into theDataA[3]["FirstName"]
    put "Nederlander" into theDataA[3]["LastName"]
    put "Three Amigo" into theDataA[3]["Title"]
    put theImageFolder & "monkey.jpg" into theDataA[3]["Image URL"]

    put "Jane" into theDataA[4]["FirstName"]
    put "Blue" into theDataA[4]["LastName"]
    put "Secret Agent" into theDataA[4]["Title"]
    put theImageFolder & "monkey.jpg" into theDataA[4]["Image URL"]

    put "Jefferson" into theDataA[5]["FirstName"]
    put "Blue" into theDataA[5]["LastName"]
    put "Secret Agent" into theDataA[5]["Title"]
    put theImageFolder & "monkey.jpg" into theDataA[5]["Image URL"]

    lock screen
    set the dgData of group "DataGrid 1" to theDataA

    ## Hilite first row
    set the dgHilitedLines of group "DataGrid 1" to 1
    unlock screen
end uiPopulatePeople
```

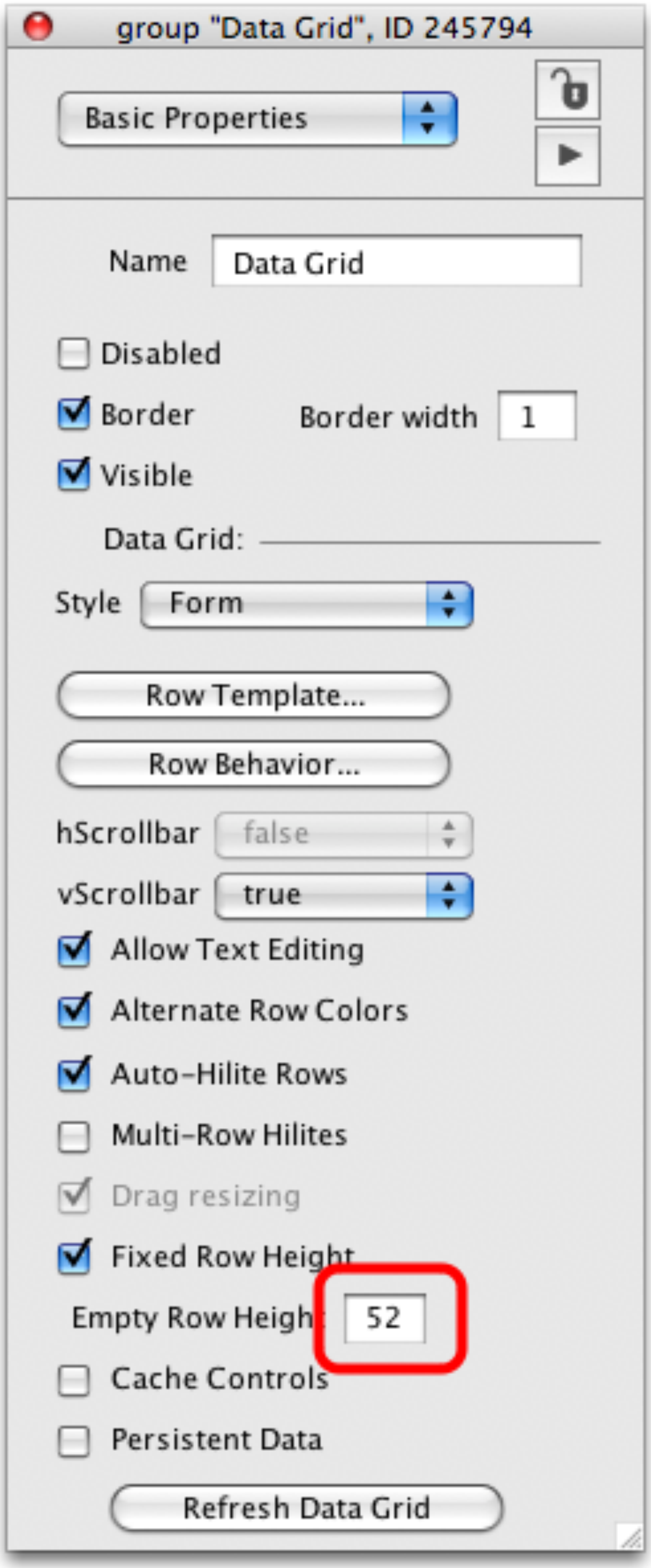
## The Result



After calling the uiPopulatePeople command the Data Grid should look something like this (1).

Once you have set the dgData property you can refresh the data grid to see any changes you might make to the row template. You can send the **ResetList** message to the data grid, or press the **Refresh Data Grid** button on the Basic Properties pane of the property inspector (2) to redraw the data using the updated template.

## Troubleshooting: Row Height


group "Data Grid", ID 245794

Basic Properties

Name: Data Grid

[ ] Disabled
[✓] Border — Border width: 1
[✓] Visible

Data Grid:

Style: Form

Row Template...
Row Behavior...

hScrollbar: false
vScrollbar: true

[✓] Allow Text Editing
[✓] Alternate Row Colors
[✓] Auto-Hilite Rows
[ ] Multi-Row Hilites
[✓] Drag resizing
[✓] Fixed Row Height
Empty Row Height: 52
[ ] Cache Controls
[ ] Persistent Data

Refresh Data Grid

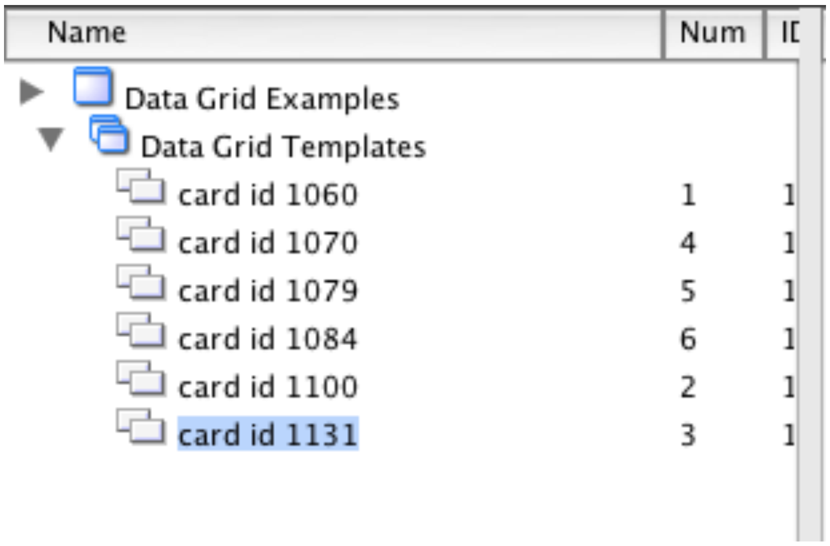
By default a data grid uses a fixed height for each row. If you have not set the row height the first time you populate the data grid with data then the value is automatically filled in based on the height of your row template. If you then change the height of the row template later on you will need to update the row height in the property inspector.

# Data Grid Fundamentals

## What Is This "Data Grid Templates" Stack That Appeared in My Project?

A data grid relies on record templates (forms) and column templates (tables) to display data. These templates are merely Revolution controls that will be used to visualize your data. In order to manage these controls the Revolution IDE creates a stack named **Data Grid Templates** the first time you add a data grid to a card in your stack.

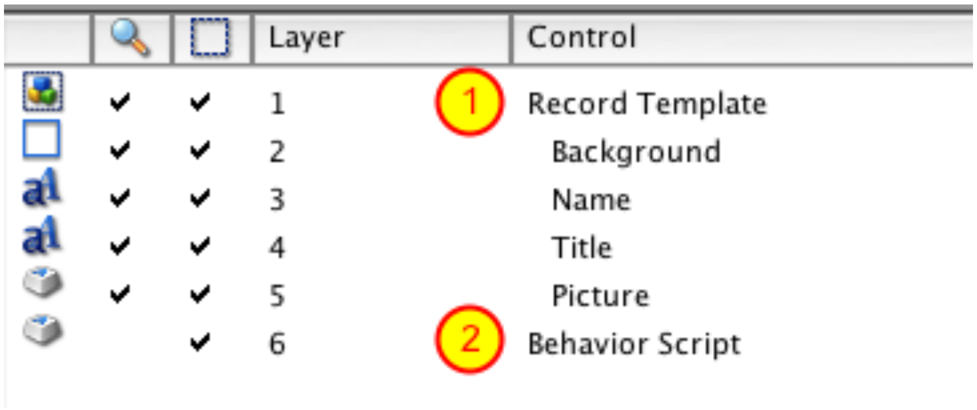
### Data Grid Templates Stack



| Name | Num | ID |
|---|---|---|
| Data Grid Examples | | |
| Data Grid Templates | | |
| card id 1060 | 1 | 1 |
| card id 1070 | 4 | 1 |
| card id 1079 | 5 | 1 |
| card id 1084 | 6 | 1 |
| card id 1100 | 2 | 1 |
| card id 1131 | 3 | 1 |

This stack contains a single card for each data grid you add to a stack. Each card contains the template(s) and behaviors that the data grid will use to display the data.

### Data Grid Templates Card Contents

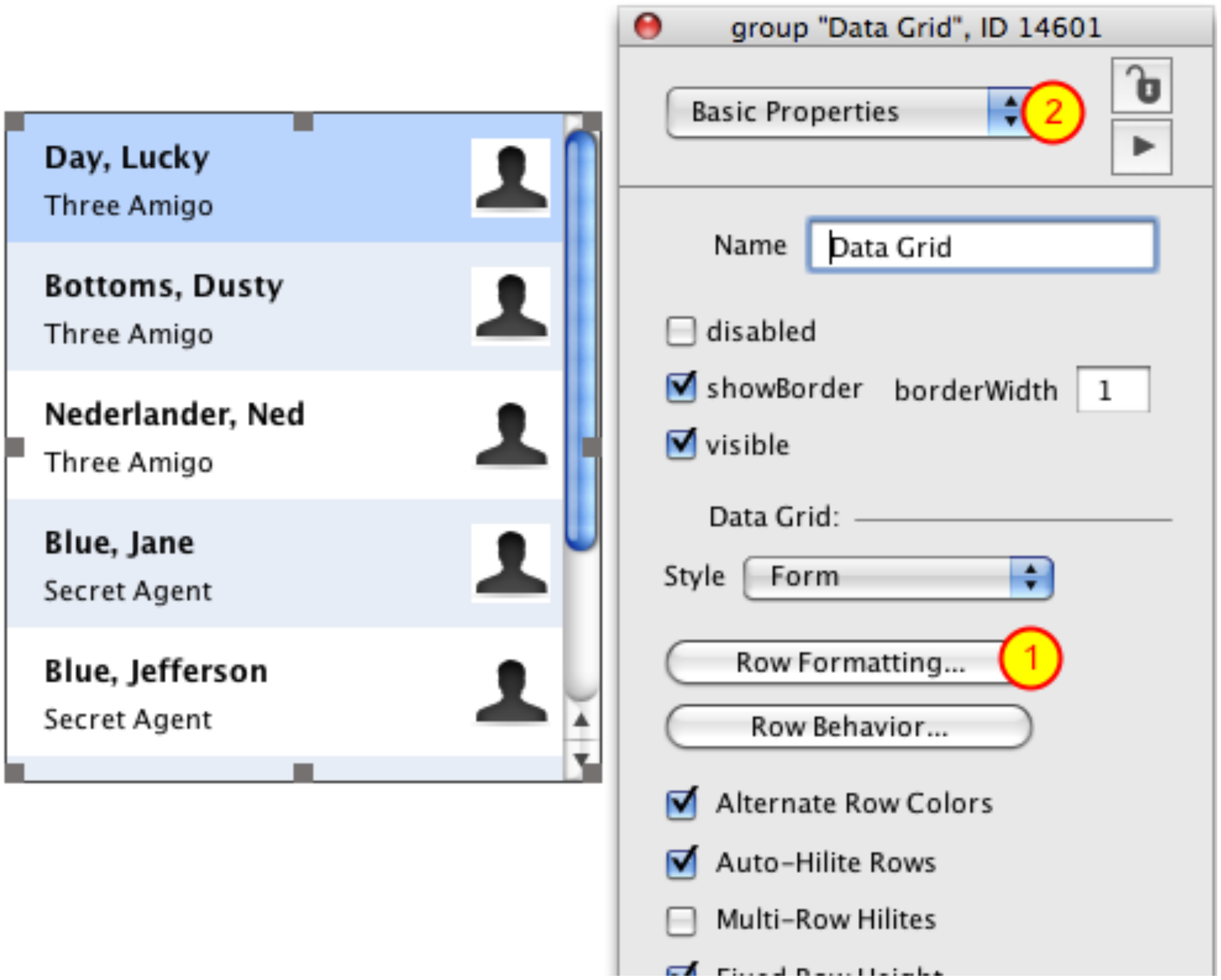


| | | | Layer | Control |
|---|---|---|---|---|
| | ✓ | ✓ | 1 | 1 Record Template |
| | ✓ | ✓ | 2 | Background |
| | ✓ | ✓ | 3 | Name |
| | ✓ | ✓ | 4 | Title |
| | ✓ | ✓ | 5 | Picture |
| | | ✓ | 6 | 2 Behavior Script |

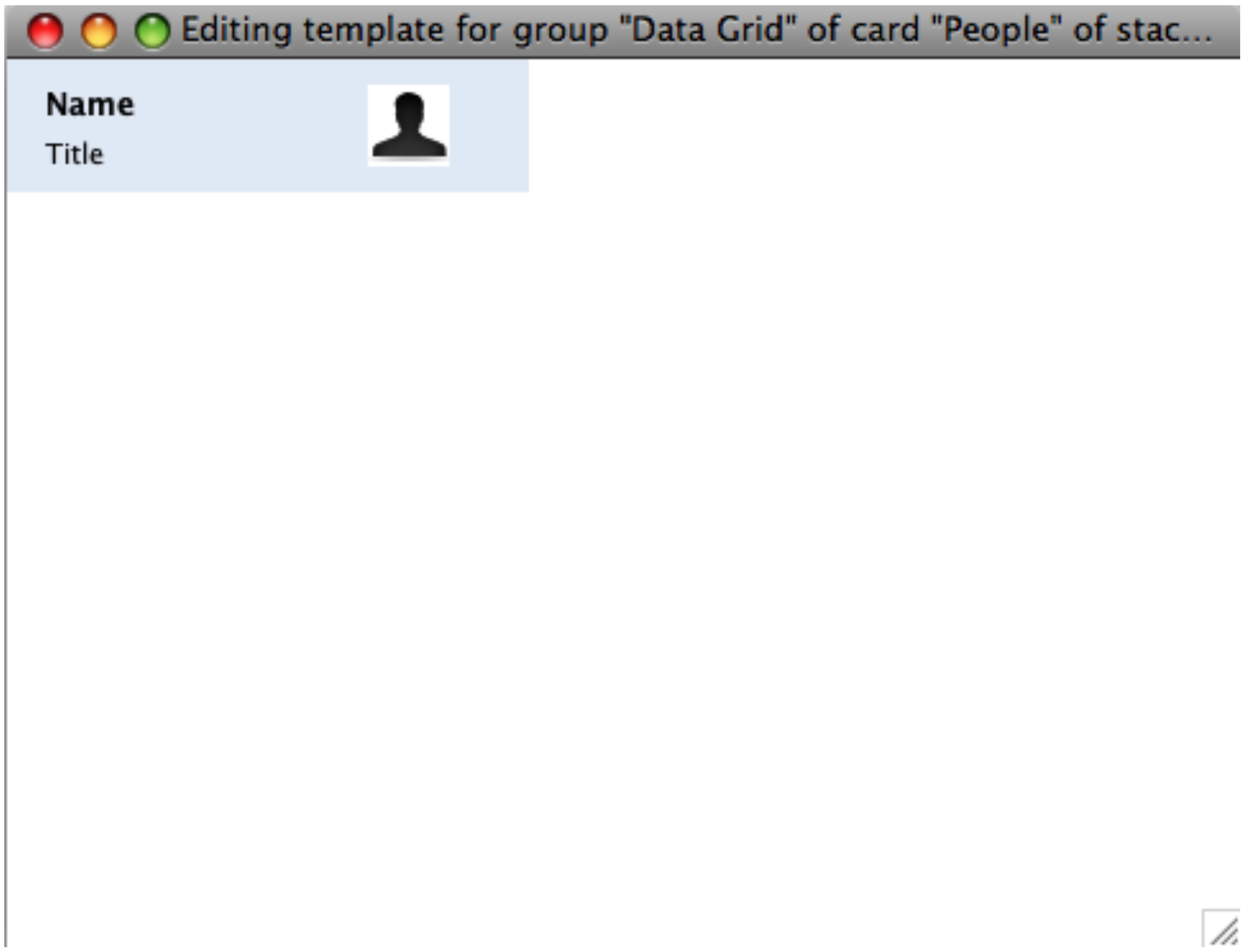
This screenshot shows some controls that might appear on a Data Grid Templates card. The **Row Template** (1) is the group that contains a) all of the controls for a form or b) the **column templates** for a table. The IDE assigns the **Row Template** group to the *dgProps["row template"]* property of 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



The Revolution IDE makes it easy to locate your templates in the Data Grid Templates stack. Simply select the data grid control and click the **Row Formatting...** button (1) in the Basic Properties (2) pane of the inspector.

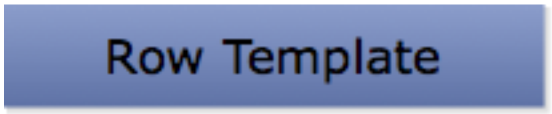

Editing template for group "Data Grid" of card "People" of stac...
Name
Title

The card containing the template(s) for the selected data grid will open.

## What is a Row Template?

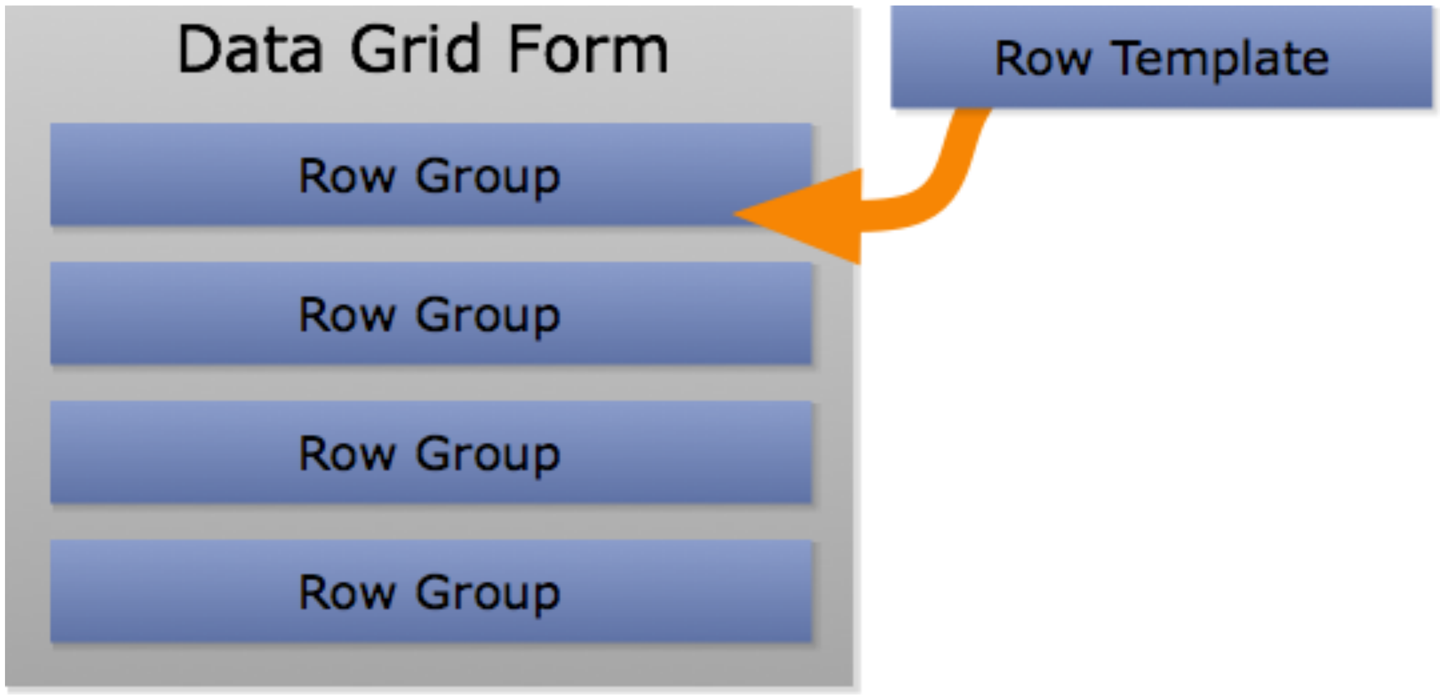
The reason a data grid can be customized is because it uses "templates" to represent data. A template is merely a Revolution group that will be copied and used to draw your data on the screen. When working with data grid forms we refer to this group as a "Row Template".

### Row Template



The **Row Template** is a Revolution group control. This group represents a single record in the data that you are displaying in a data grid form. This group can contain any Revolution control and the look and layout are entirely controlled by you.

### How A Record Template Is Used

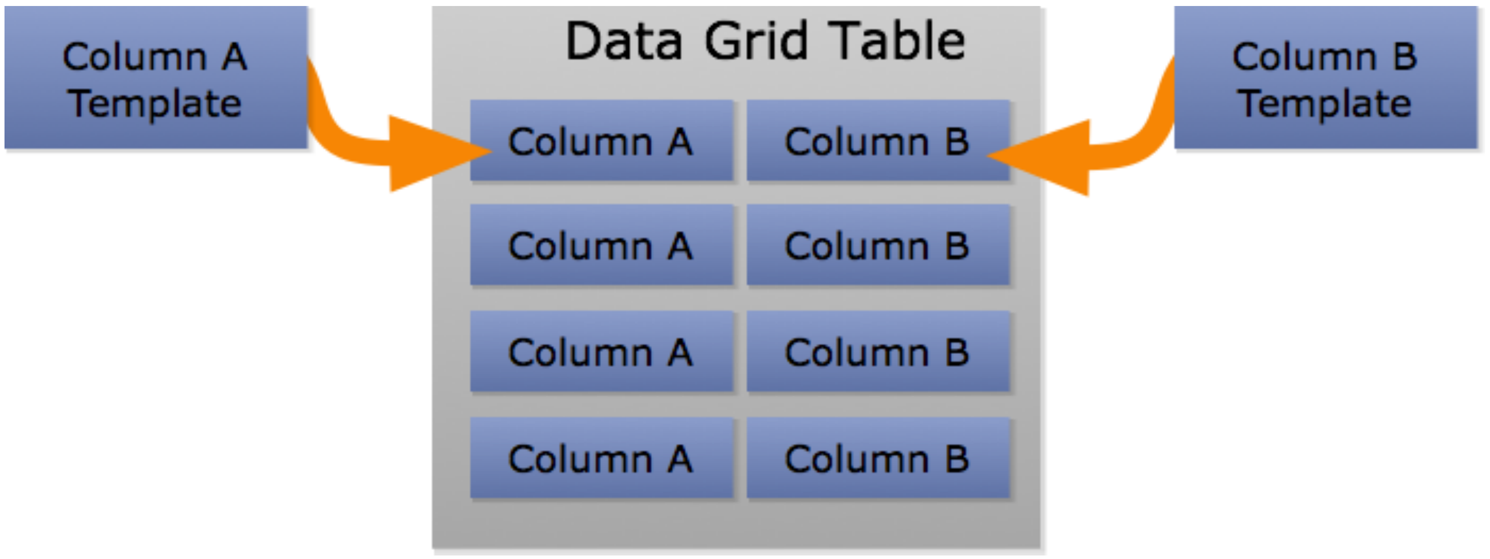


When the data grid form displays your data it copies the **Row Template** into itself. It makes just enough copies to cover the visible area and then inserts data into these copies of the **Row Template**. As the user uses the scrollbar to scroll through the data the same copies are used but new data is inserted. This means that the data grid form is never drawing more records than the user can actually see which means the data grid is fast.

## What is a Column Template?

Like data grid forms, data grid tables also use templates. The difference is that when working with a table you create a template to represent each column rather than a template to represent each row. We refer to these templates as "Column Templates". A column template is similar to a record template. One difference is that a column template is not limited to being a group. You can use a field for a template, a graphic, a button, etc.

### How A Column Template Is Used

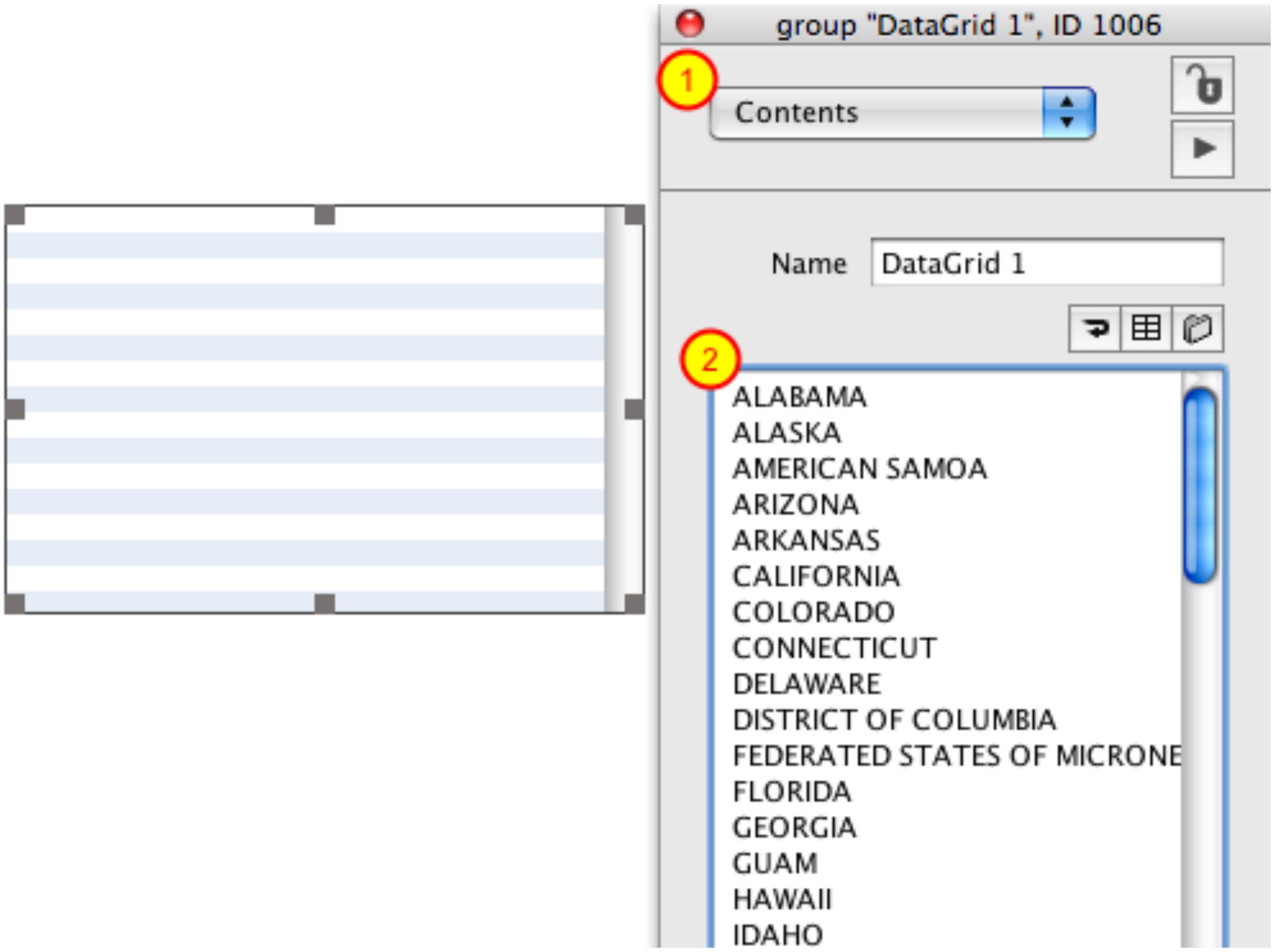


When your data is drawn into a data grid table each column template is copied into the data grid as many times as necessary in order to fill the visible area. Data is then inserted into these templates as the user scrolls.

## How Do I Populate a Data Grid With Data?

There are a couple of ways you assign data to a data grid. This lesson will show you how.

### Use The Property Inspector



group "DataGrid 1", ID 1006

Name: DataGrid 1

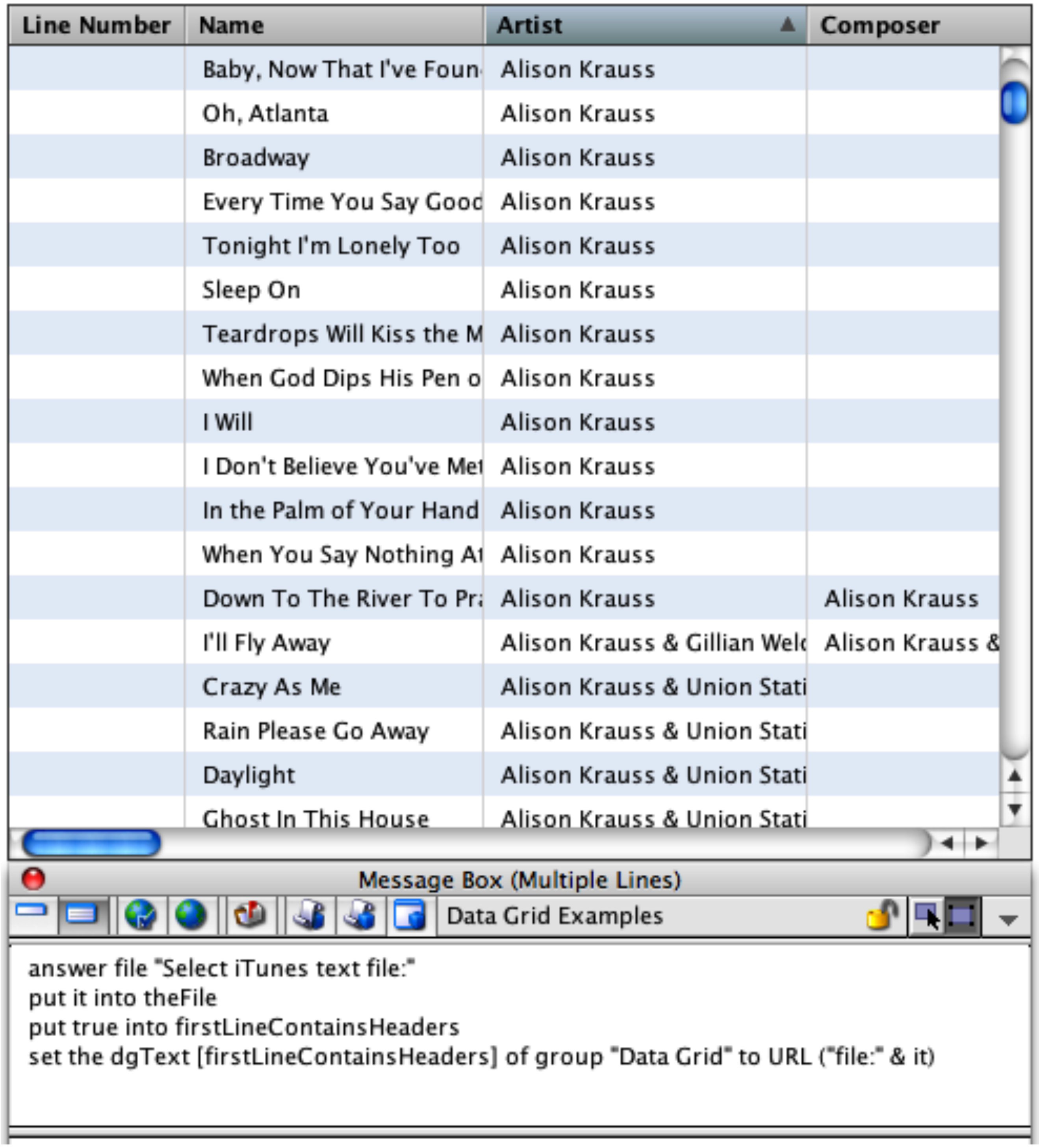
ALABAMA
ALASKA
AMERICAN SAMOA
ARIZONA
ARKANSAS
CALIFORNIA
COLORADO
CONNECTICUT
DELAWARE
DISTRICT OF COLUMBIA
FEDERATED STATES OF MICRONE
FLORIDA
GEORGIA
GUAM
HAWAII
IDAHO

You can use the Contents pane of the property inspector to quickly assign data to a data grid. This does not provide as many options as setting the properties mentioned below but it will get you up and running quickly.

1) Switch to the **Contents** pane of the property inspector.
2) Enter some tab delimited text into the field.

When you use the property inspector to assign data to the data grid the property inspector sets the dgText property of the data grid.

## Set the dgText Property



| Line Number | Name | Artist | Composer |
|---|---|---|---|
| | Baby, Now That I've Foun | Alison Krauss | |
| | Oh, Atlanta | Alison Krauss | |
| | Broadway | Alison Krauss | |
| | Every Time You Say Gooc | Alison Krauss | |
| | Tonight I'm Lonely Too | Alison Krauss | |
| | Sleep On | Alison Krauss | |
| | Teardrops Will Kiss the M | Alison Krauss | |
| | When God Dips His Pen o | Alison Krauss | |
| | I Will | Alison Krauss | |
| | I Don't Believe You've Met | Alison Krauss | |
| | In the Palm of Your Hand | Alison Krauss | |
| | When You Say Nothing At | Alison Krauss | |
| | Down To The River To Pr | Alison Krauss | Alison Krauss |
| | I'll Fly Away | Alison Krauss & Gillian Welc | Alison Krauss & |
| | Crazy As Me | Alison Krauss & Union Stati | |
| | Rain Please Go Away | Alison Krauss & Union Stati | |
| | Daylight | Alison Krauss & Union Stati | |
| | Ghost In This House | Alison Krauss & Union Stati | |

```
answer file "Select iTunes text file:"
put it into theFile
put true into firstLineContainsHeaders
set the dgText [firstLineContainsHeaders] of group "Data Grid" to URL ("file:" & it)
```

The data grid works with arrays behind the scenes but in the interest of making life easier for some folks there is a dgText property. The syntax is as follows:

*set the dgText [ pFirstLineContainsHeaders ] of group "Data Grid" to pText*

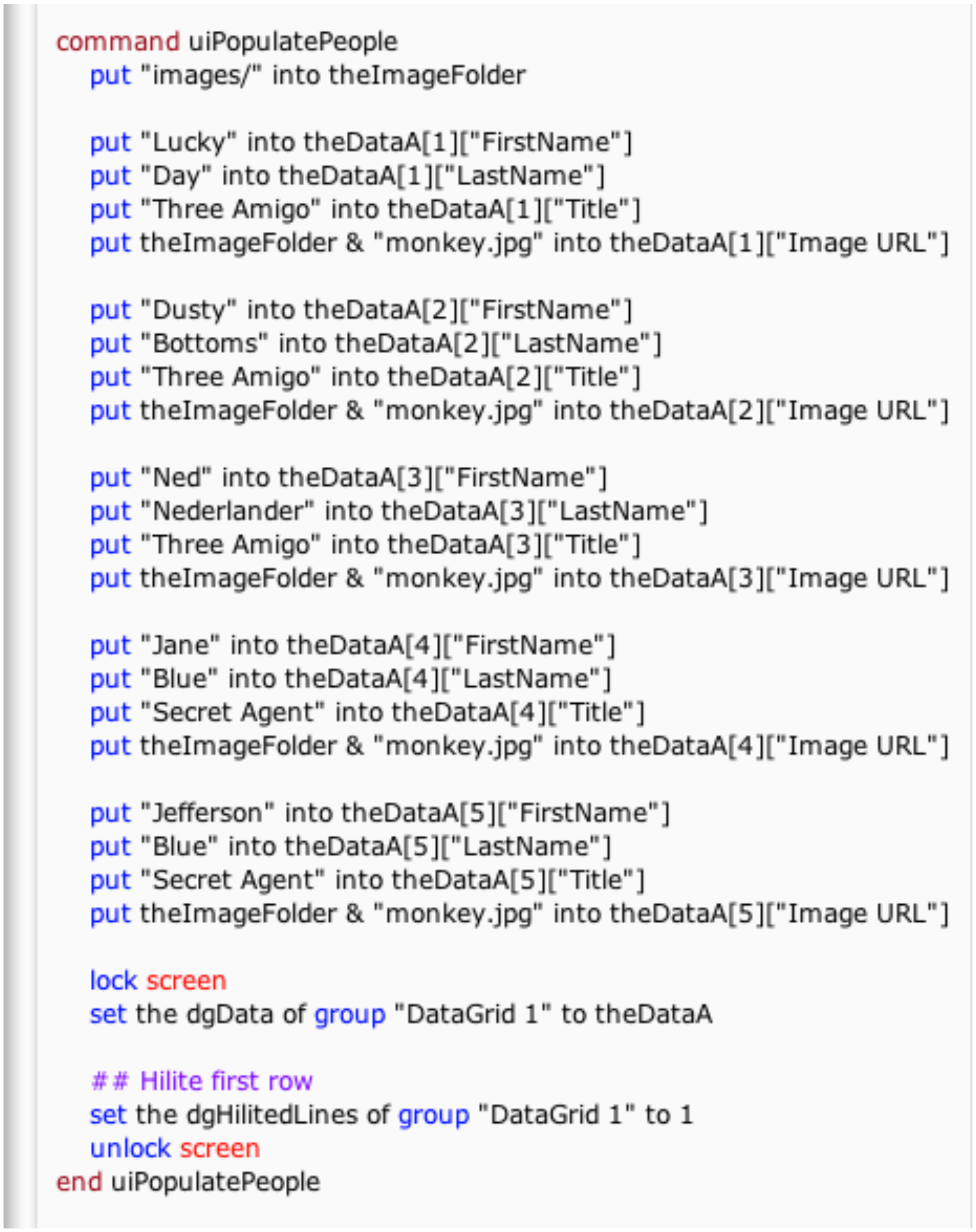
*pText* is assumed to be a collection of data where each row is delimited by the return character and each item is delimited by a tab. You can map each item of each line in *pText* to a particular key in an array (and thus a table column) by passing in true for *pFirstLineContainsHeaders*. If true then the data grid will extract the first line of *pText* and use the values for the internal key/column names. The default value for *pFirstLineContainsHeaders* is false.

If you set the dgText of a **data grid table** then all data will be imported and assigned to the appropriate column depending on the value of *pFirstLineContainsHeaders.* Normally you should set this property to true and provide the header that maps each item of each line to a specific column. Note that if *pFirstLineContainsHeaders* is true then the columns must already exist in your data grid table in order to be displayed.

If *pFirstLineContainsHeaders* is false then the *columns* property of the data grid is used for mapping. For example, the first item of a line of *pText* would be assigned to the column that appears on the first line in the *columns* property of the data grid. If line 1 of *pText* contains more items than there are columns in the table then new columns are added. Any new columns are named "Col 1", "Col 2", etc.

If you set the dgText property of a **data grid form** then the data will be imported but it is up to you to modify your **Row Template Behavior** to display the imported data correctly. If *pFirstLineContainsHeaders* is false then each item of each line in *pText* will be named "Label X" (where X is the item number) in the array that is passed to FillInData.

## Set the dgData Property



```
command uiPopulatePeople
   put "images/" into theImageFolder

   put "Lucky" into theDataA[1]["FirstName"]
   put "Day" into theDataA[1]["LastName"]
   put "Three Amigo" into theDataA[1]["Title"]
   put theImageFolder & "monkey.jpg" into theDataA[1]["Image URL"]

   put "Dusty" into theDataA[2]["FirstName"]
   put "Bottoms" into theDataA[2]["LastName"]
   put "Three Amigo" into theDataA[2]["Title"]
   put theImageFolder & "monkey.jpg" into theDataA[2]["Image URL"]

   put "Ned" into theDataA[3]["FirstName"]
   put "Nederlander" into theDataA[3]["LastName"]
   put "Three Amigo" into theDataA[3]["Title"]
   put theImageFolder & "monkey.jpg" into theDataA[3]["Image URL"]

   put "Jane" into theDataA[4]["FirstName"]
   put "Blue" into theDataA[4]["LastName"]
   put "Secret Agent" into theDataA[4]["Title"]
   put theImageFolder & "monkey.jpg" into theDataA[4]["Image URL"]

   put "Jefferson" into theDataA[5]["FirstName"]
   put "Blue" into theDataA[5]["LastName"]
   put "Secret Agent" into theDataA[5]["Title"]
   put theImageFolder & "monkey.jpg" into theDataA[5]["Image URL"]

   lock screen
   set the dgData of group "DataGrid 1" to theDataA

   ## Hilite first row
   set the dgHilitedLines of group "DataGrid 1" to 1
   unlock screen
end uiPopulatePeople
```

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. Here is an example with five records:

```
theDataA[1]
theDataA[2]
theDataA[3]
theDataA[4]
theDataA[5]
```

You then store all of your custom data in the second dimension. The following is how you would display five records, each with a "Name" property:

```
theDataA[1]["Name"]
theDataA[2]["Name"]
theDataA[3]["Name"]
theDataA[4]["Name"]
theDataA[5]["Name"]
```

In the example code above there are 5 records, each with a "FirstName", "LastName" and "Title" property. Your templates will have access to these properties when the time comes to draw the data on the screen.

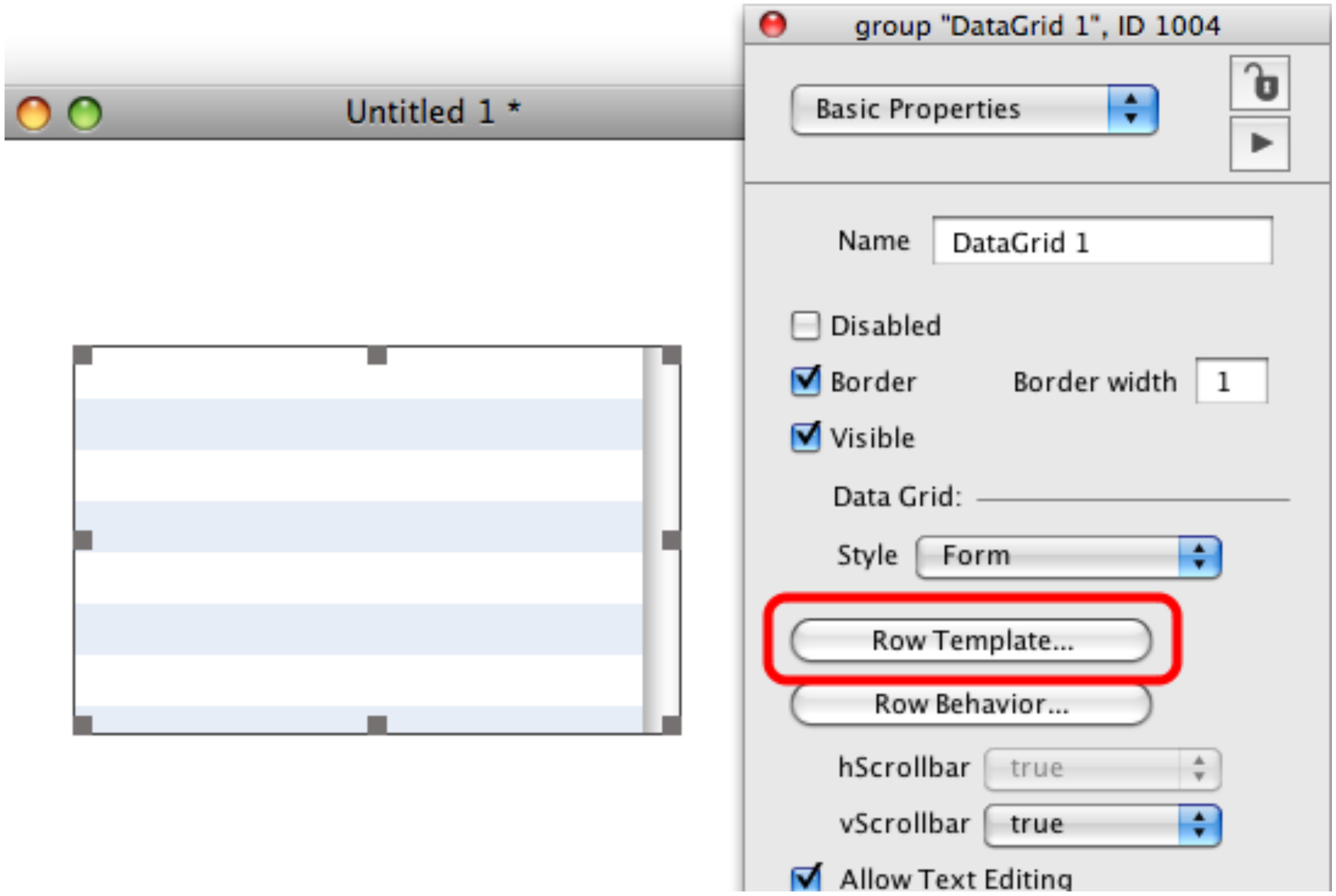
**Important:** if you are using a data grid table then a key in the array must match the name of a column in the table in order to be displayed. So if you have a column named "Image URL" in your table you must have a corresponding key named "Image URL":

```
theDataA[1]["Image URL"]
```

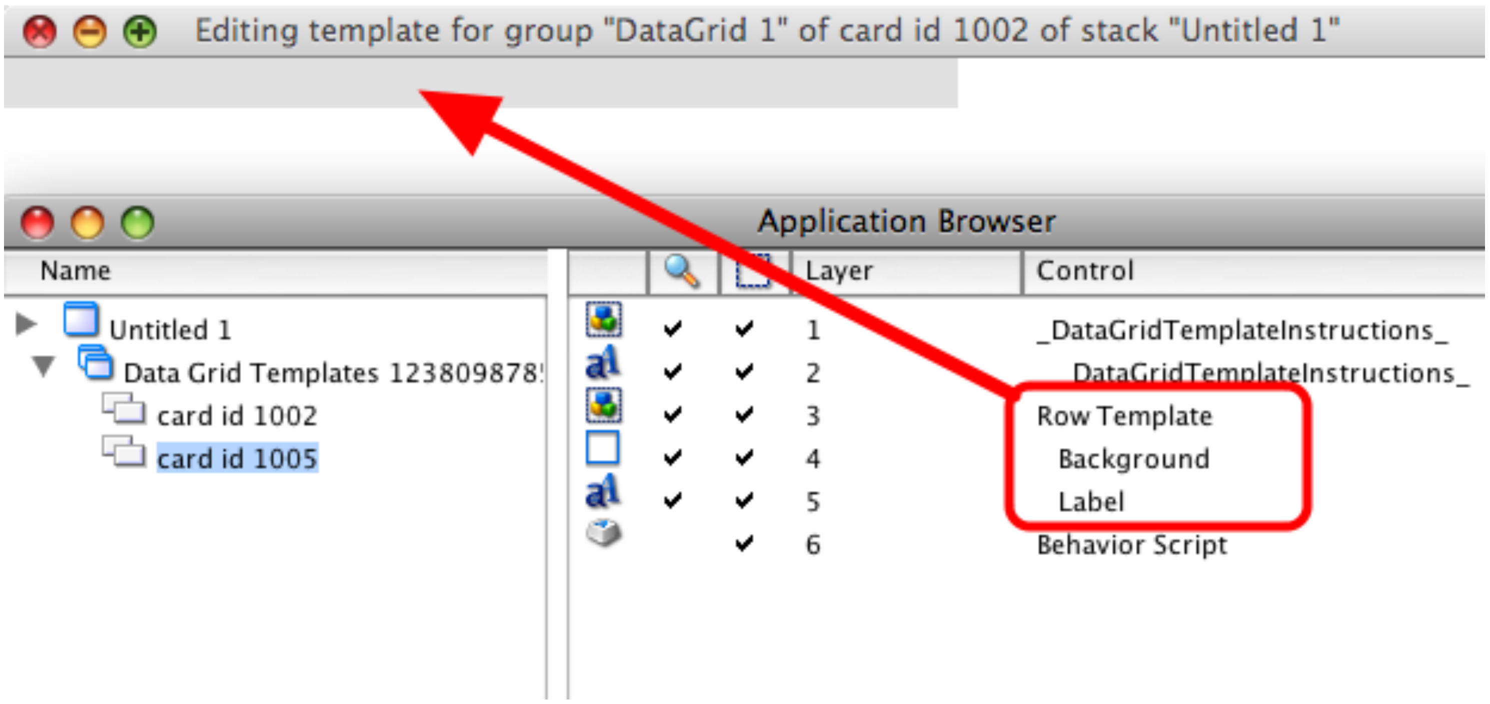
## How Do I Customize A Form's Row Template?

When you drop a data grid from the Tools palette onto a card a record template is created for you automatically.  This template will display line delimited text but you will most likely want to customize the template. This lesson will show you how to begin customizing the template to meet your needs.

### Reveal the Row Template



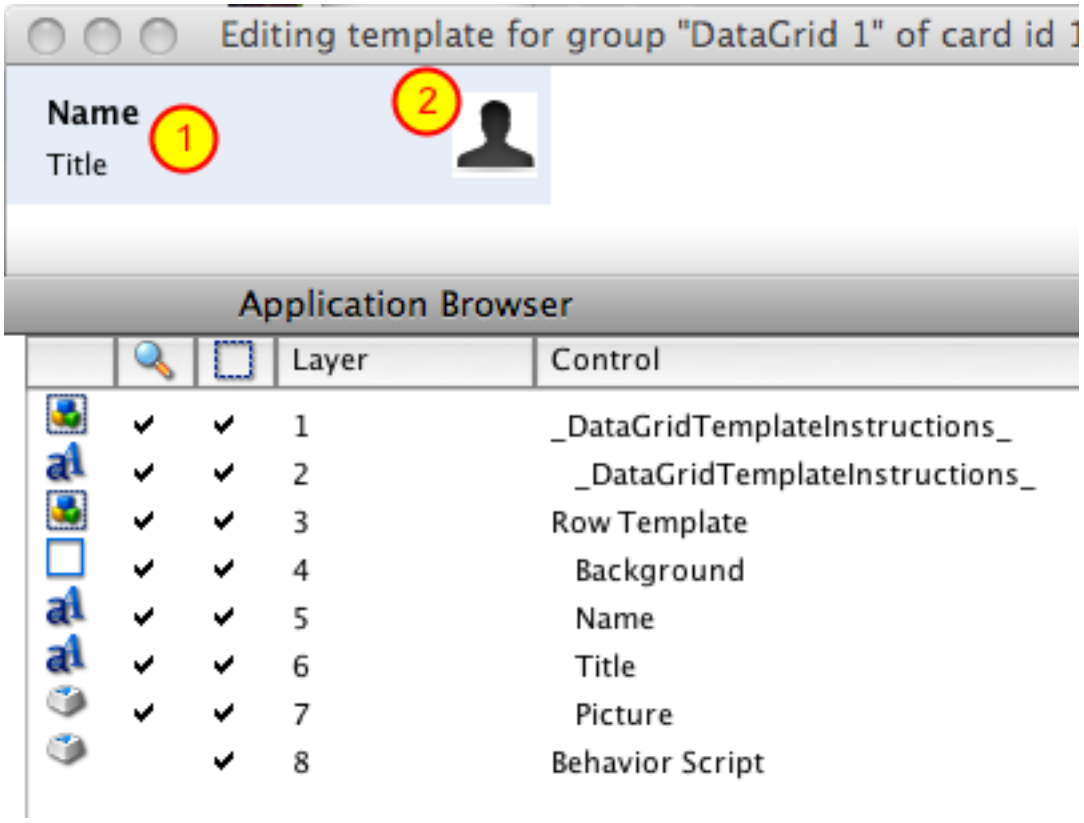
Select the data grid and open the property inspector. Click on the **Row Template...** button to open the card that the data grid's template is on.


Editing template for group "DataGrid 1" of card id 1002 of stack "Untitled 1"
Application Browser
Name
Untitled 1
Data Grid Templates 123809878!
card id 1002
card id 1005

| | | | Layer | Control |
|---|---|---|---|---|
| | ✓ | ✓ | 1 | _DataGridTemplateInstructions_ |
| | ✓ | ✓ | 2 | _DataGridTemplateInstructions_ |
| | ✓ | ✓ | 3 | Row Template |
| | ✓ | ✓ | 4 | Background |
| | ✓ | ✓ | 5 | Label |
| | | ✓ | 6 | Behavior Script |

The template for the data grid will open. You can now edit the controls within the **Row Template** group in order to customize the look and feel according to your needs.

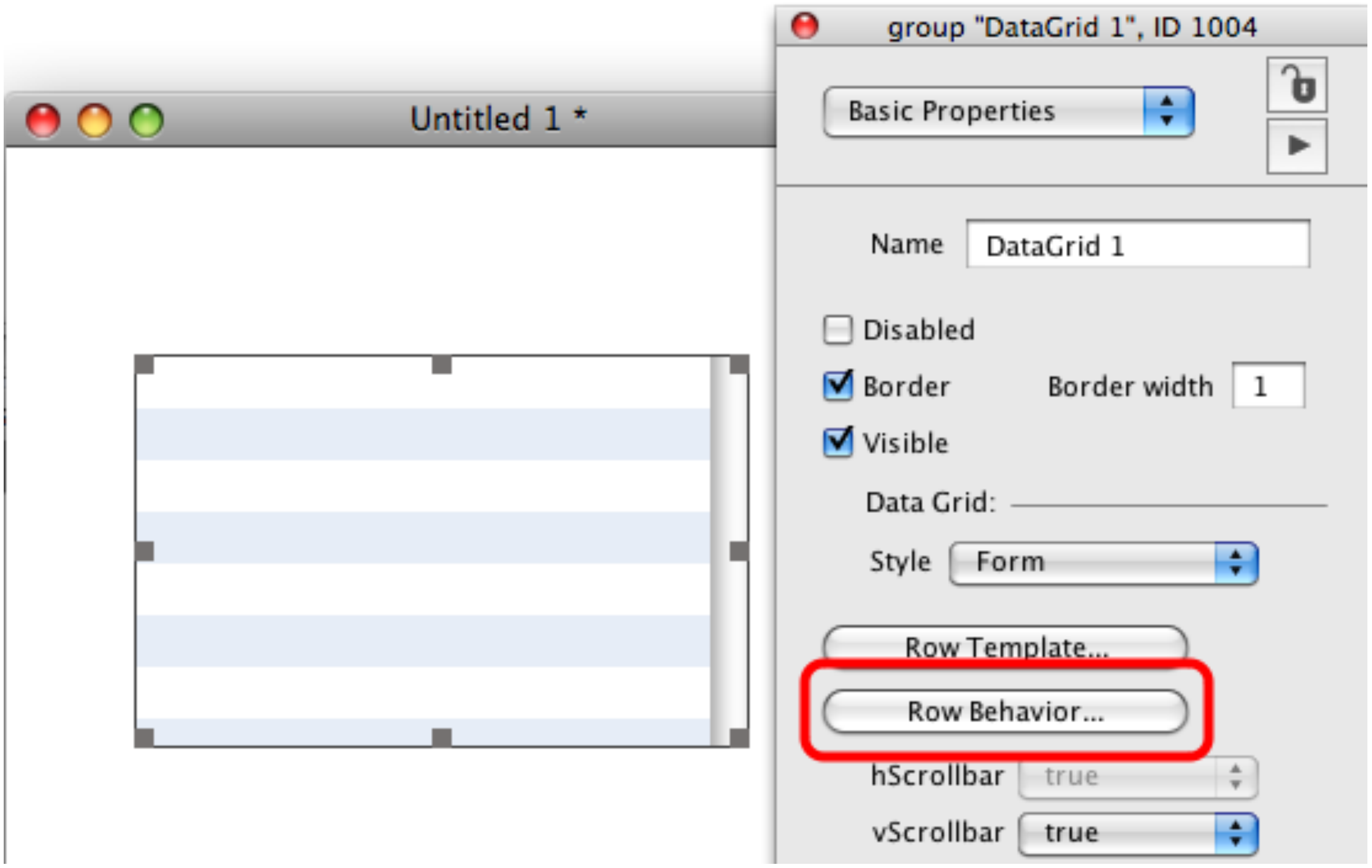
## Example Of A Customized Template



| | | | Layer | Control |
|---|---|---|---|---|
| | ✓ | ✓ | 1 | _DataGridTemplateInstructions_ |
| | ✓ | ✓ | 2 | _DataGridTemplateInstructions_ |
| | ✓ | ✓ | 3 | Row Template |
| | ✓ | ✓ | 4 | Background |
| | ✓ | ✓ | 5 | Name |
| | ✓ | ✓ | 6 | Title |
| | ✓ | ✓ | 7 | Picture |
| | | ✓ | 8 | Behavior Script |

Here is an example of a customized template. I've added a Name and Title field (1) as well as a control for displaying a picture of the person (2).

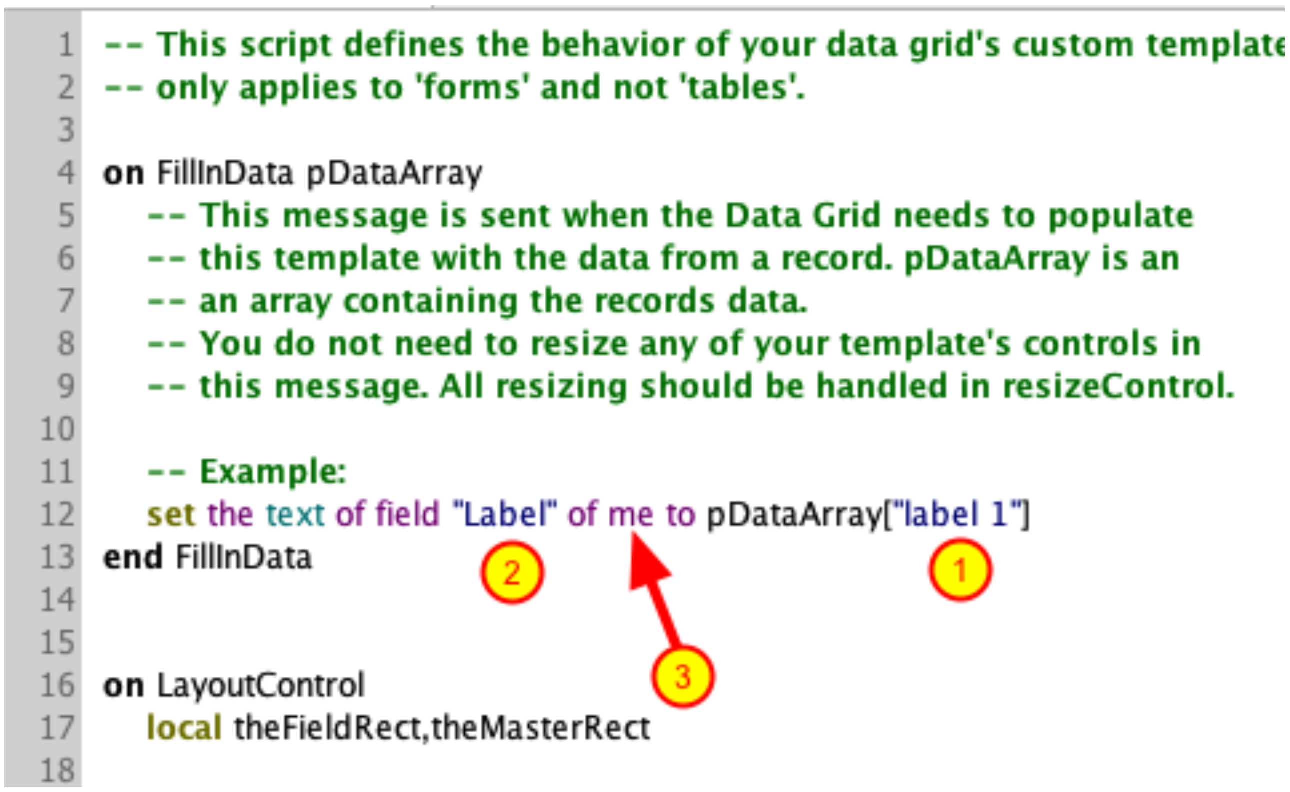
## Edit Behavior



After customizing the controls in our template you need to update the **behavior** associated with the template. The behavior script is where you tell the data grid what data goes in which controls as well as how to position the controls.

Click the **Row Behavior...** button to open the behavior script.

## Edit the Behavior Script



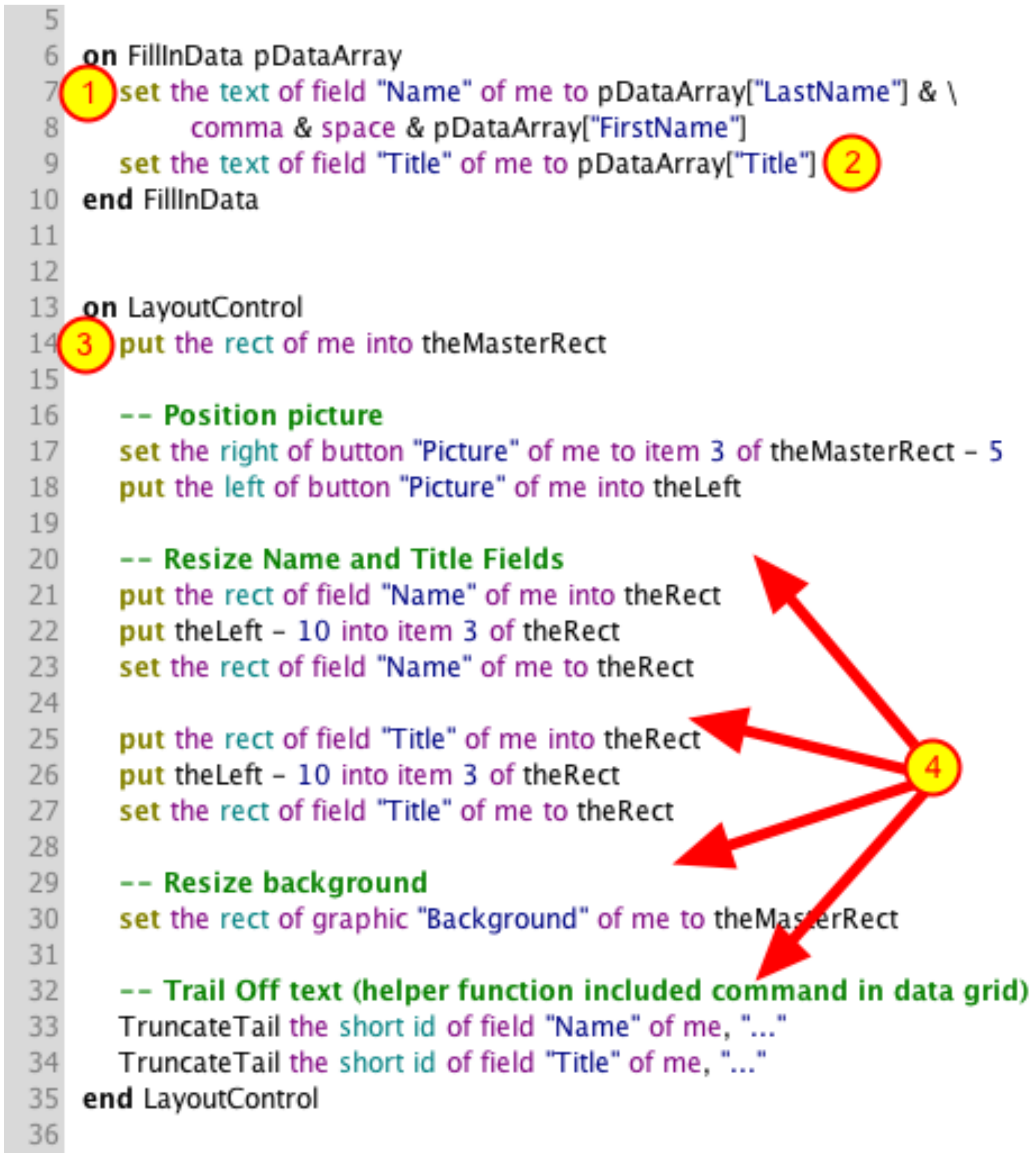
```
-- This script defines the behavior of your data grid's custom template
-- only applies to 'forms' and not 'tables'.

on FillInData pDataArray
   -- This message is sent when the Data Grid needs to populate
   -- this template with the data from a record. pDataArray is an
   -- an array containing the records data.
   -- You do not need to resize any of your template's controls in
   -- this message. All resizing should be handled in resizeControl.

   -- Example:
   set the text of field "Label" of me to pDataArray["label 1"]
end FillInData


on LayoutControl
   local theFieldRect,theMasterRect

```

The default behavior script contains comments explaining what to do in each handler. The **FillInData** message is where you move data from a record into a Revolution control. To do this you assign keys of pDataArray (1) to a control (2).

Note that the first parameter passed to **FillInData** is an array representing the current row being displayed. Also note how this is different than the first parameter passed to **FillInData** for a **column template** which is the value of the column.

**Important:** Make sure that you always refer to controls in the template using 'of me' (3). Since multiple copies of the template will be made you will have multiple controls with the same names. Using 'of me' removes any ambiguity and ensures that data is displayed in the correct control.

## An Example of a Customized Behavior



```
on FillInData pDataArray
   set the text of field "Name" of me to pDataArray["LastName"] & \
         comma & space & pDataArray["FirstName"]
   set the text of field "Title" of me to pDataArray["Title"]
end FillInData


on LayoutControl
   put the rect of me into theMasterRect

   -- Position picture
   set the right of button "Picture" of me to item 3 of theMasterRect - 5
   put the left of button "Picture" of me into theLeft

   -- Resize Name and Title Fields
   put the rect of field "Name" of me into theRect
   put theLeft - 10 into item 3 of theRect
   set the rect of field "Name" of me to theRect

   put the rect of field "Title" of me into theRect
   put theLeft - 10 into item 3 of theRect
   set the rect of field "Title" of me to theRect

   -- Resize background
   set the rect of graphic "Background" of me to theMasterRect

   -- Trail Off text (helper function included command in data grid)
   TruncateTail the short id of field "Name" of me, "..."
   TruncateTail the short id of field "Title" of me, "..."
end LayoutControl
```

This is an example of a behavior that has been customized. The *dgData* of the data grid this template is associated with looks like this:

theDataA[1]["FirstName"]
theDataA[1]["LastName"]
theDataA[1]["Title"]

theDataA[2]["FirstName"]
theDataA[2]["LastName"]

```
theDataA[2]["Title"]

...
```

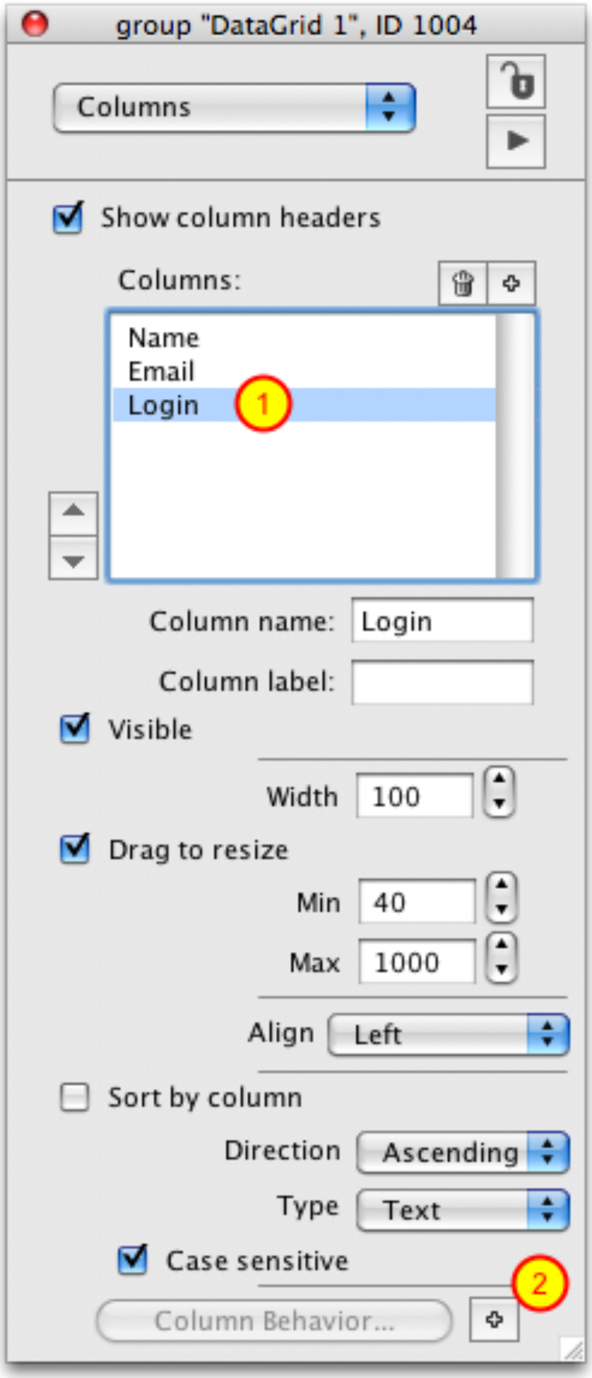
When filling in the data I place the "FirstName" and "LastName" values into a "Name" field (1) and the "Title" value into a "Title" field (2).

This example also shows how to use **LayoutControl** to layout your row template. Notice how I capture the rectangle of the row template at the beginning (3) and then use it to position all of the other elements (4).

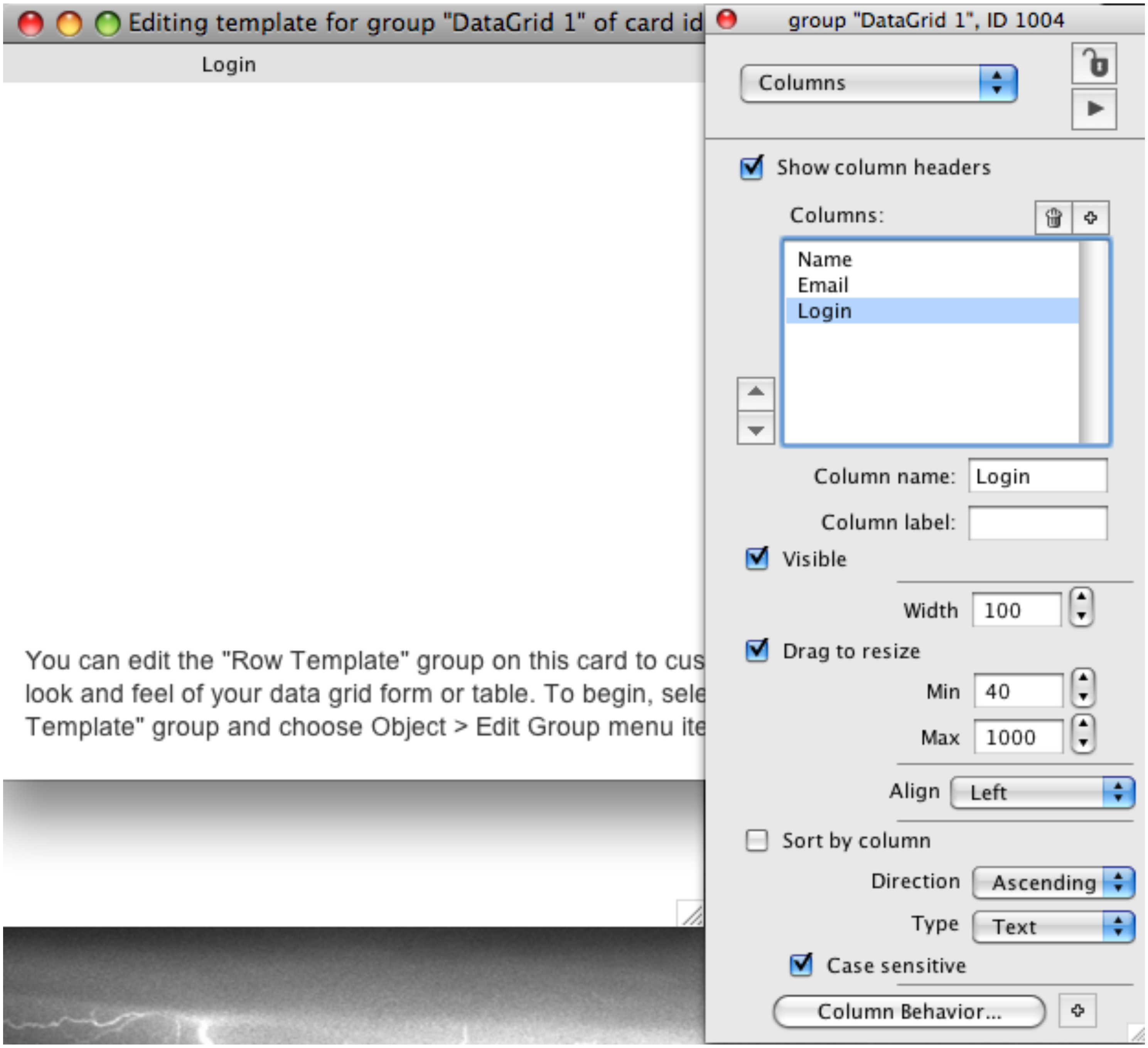
## How Do I Customize A Table's Columns?

A column template is nothing more than a Revolution control that is named after a column in your table. This control is located in the row template group for your data grid. This lesson will discuss how to create templates for columns in a data grid table.

### Use Property Inspector To Create a Column Template



In the Property Inspector, select the column you would like to customize (1). Click the "+" button at the bottom of the pane to create a column template (2).


Editing template for group "DataGrid 1" of card id

Login

You can edit the "Row Template" group on this card to cus
look and feel of your data grid form or table. To begin, sele
Template" group and choose Object > Edit Group menu ite

group "DataGrid 1", ID 1004

Columns

[✓] Show column headers

Columns:

Name
Email
Login

Column name: Login

Column label:

[✓] Visible

Width 100

[✓] Drag to resize

Min 40

Max 1000

Align Left

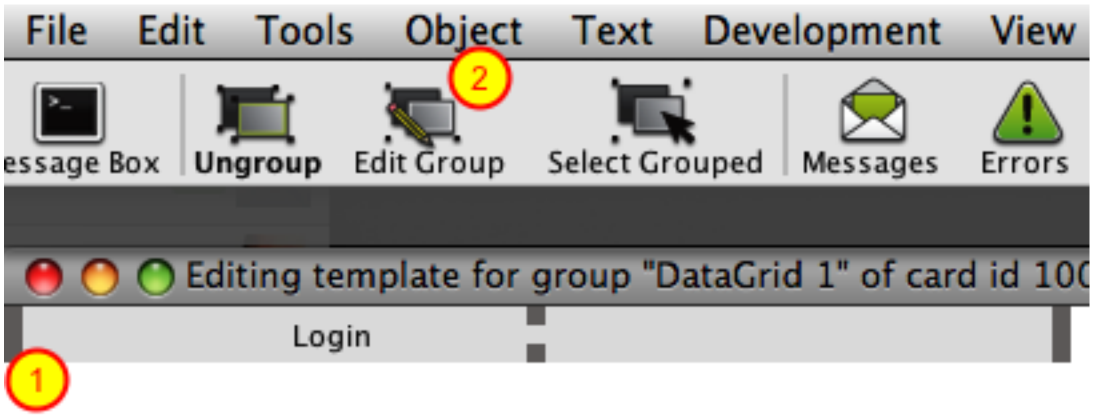
[ ] Sort by column

Direction Ascending

Type Text

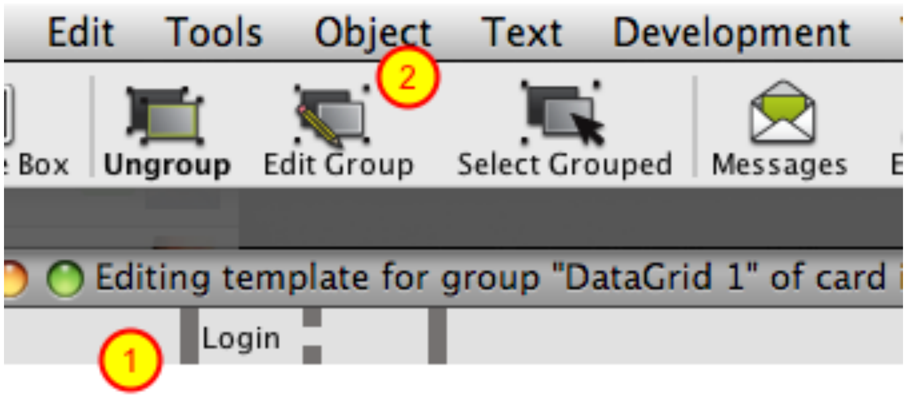
[✓] Case sensitive

## Edit the Row Template Group



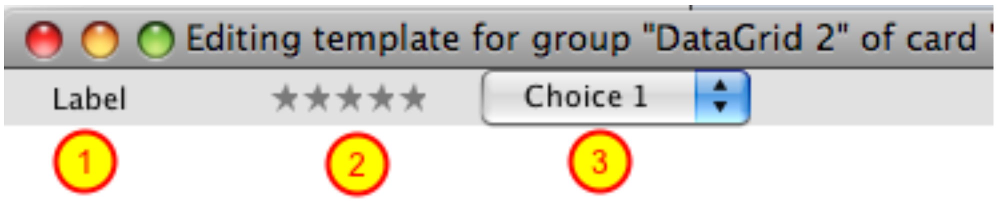
In order to customize the look we need to edit the contents of the template group. Select it (1) and click "Edit Group" (2) in the Revolution toolbar.

## Edit Column Group



Now select the column group (1) and click "Edit Group" again (2). At this point you can customize what controls appear in the column template.

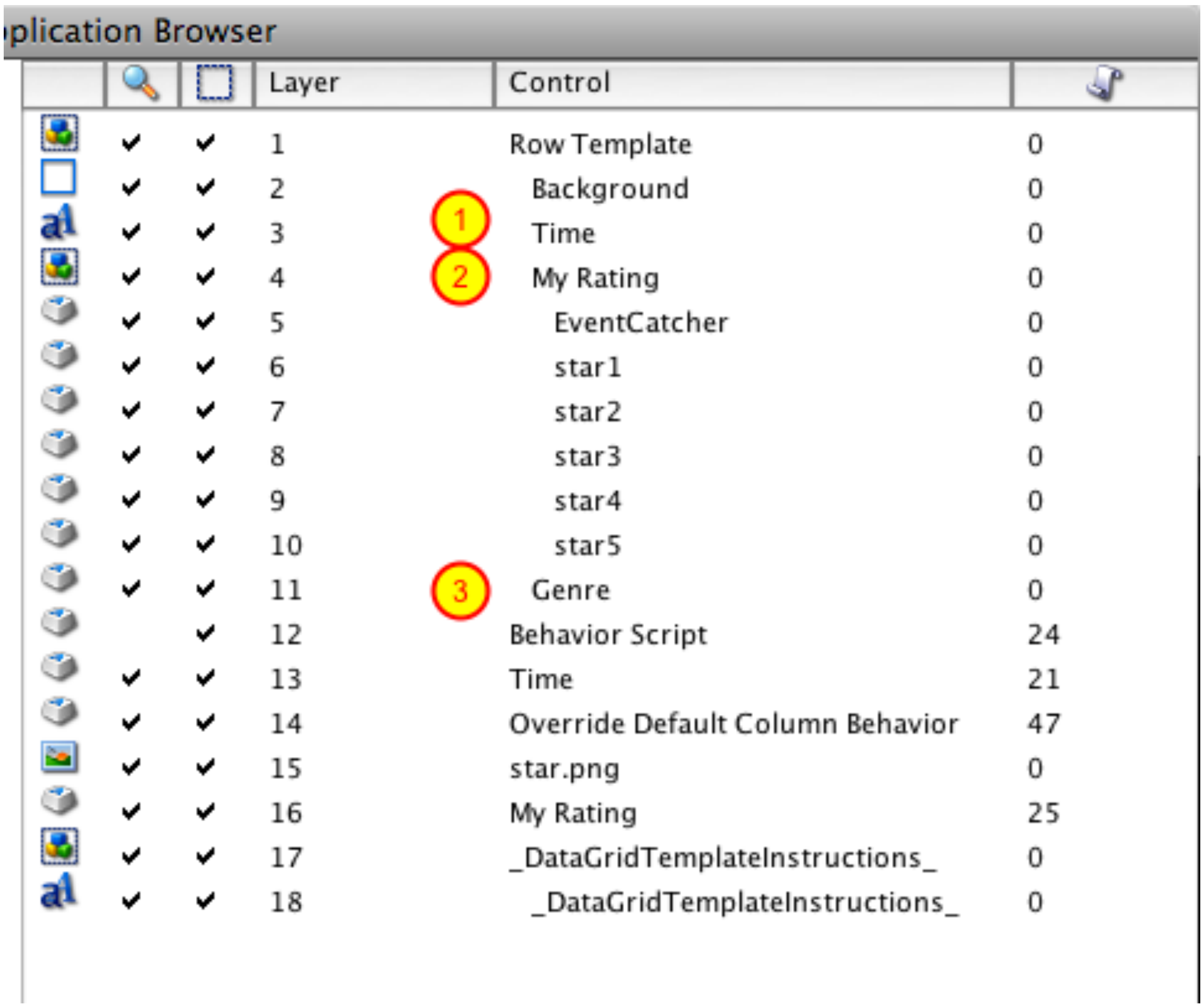
## A Column Template



In this example I have created three controls for three different columns in my data grid table.

1) A field named "Time" that formats data into a time format (I just renamed the field that was included

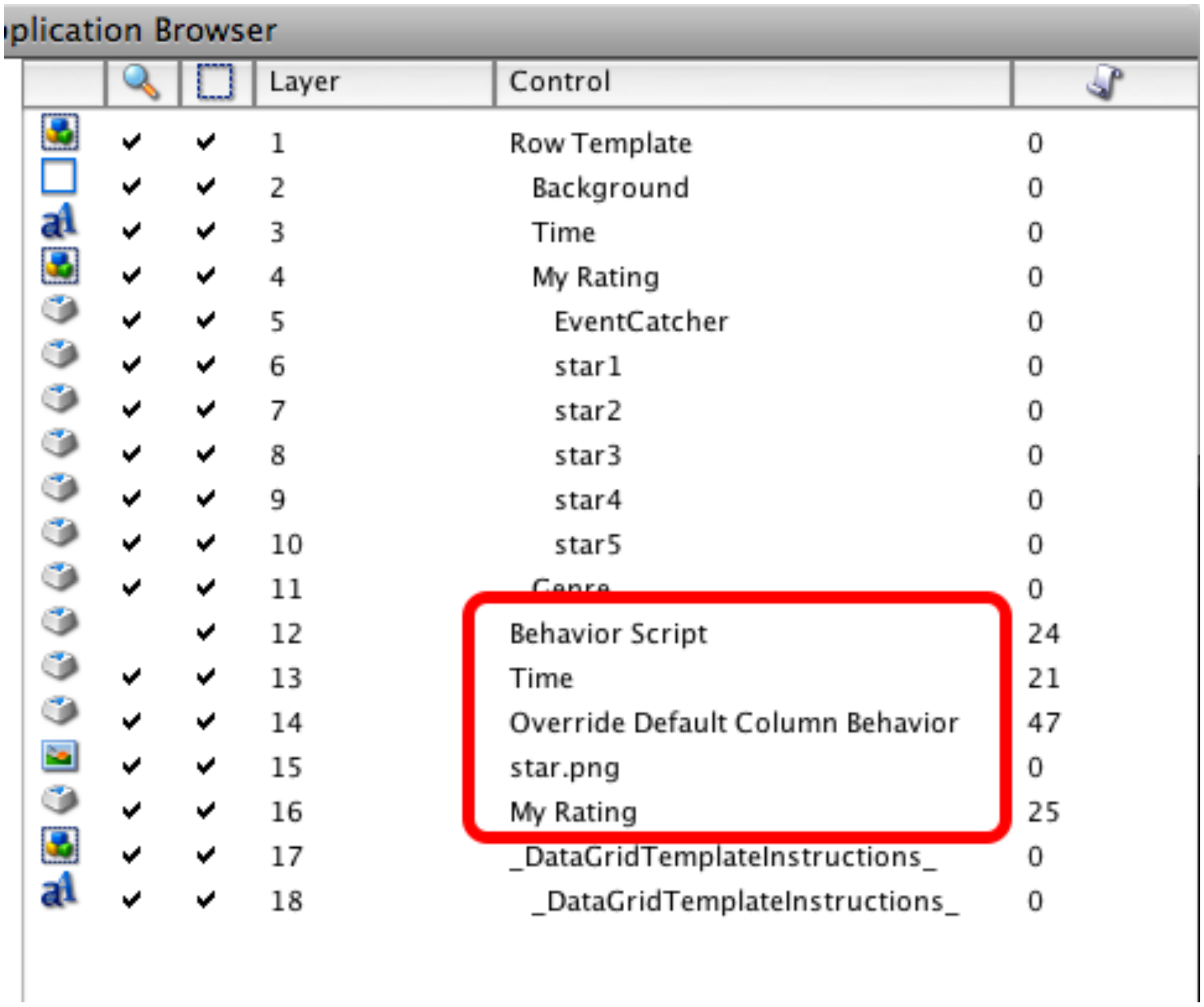
in the row template by default).
2) A group named "My Rating". The group has some buttons that display the image of a star.
3) A button named "Genre" that offers different Genre selections.



| | | | Layer | Control | |
|---|---|---|---|---|---|
| | ✓ | ✓ | 1 | Row Template | 0 |
| | ✓ | ✓ | 2 | Background | 0 |
| | ✓ | ✓ | 3 | Time | 0 |
| | ✓ | ✓ | 4 | My Rating | 0 |
| | ✓ | ✓ | 5 | EventCatcher | 0 |
| | ✓ | ✓ | 6 | star1 | 0 |
| | ✓ | ✓ | 7 | star2 | 0 |
| | ✓ | ✓ | 8 | star3 | 0 |
| | ✓ | ✓ | 9 | star4 | 0 |
| | ✓ | ✓ | 10 | star5 | 0 |
| | ✓ | ✓ | 11 | Genre | 0 |
| | | ✓ | 12 | Behavior Script | 24 |
| | ✓ | ✓ | 13 | Time | 21 |
| | ✓ | ✓ | 14 | Override Default Column Behavior | 47 |
| | ✓ | ✓ | 15 | star.png | 0 |
| | ✓ | ✓ | 16 | My Rating | 25 |
| | ✓ | ✓ | 17 | _DataGridTemplateInstructions_ | 0 |
| | ✓ | ✓ | 18 | _DataGridTemplateInstructions_ | 0 |

Here is what those controls look like in the Application Browser. Notice how all of the controls are located in the "Row Template" group.

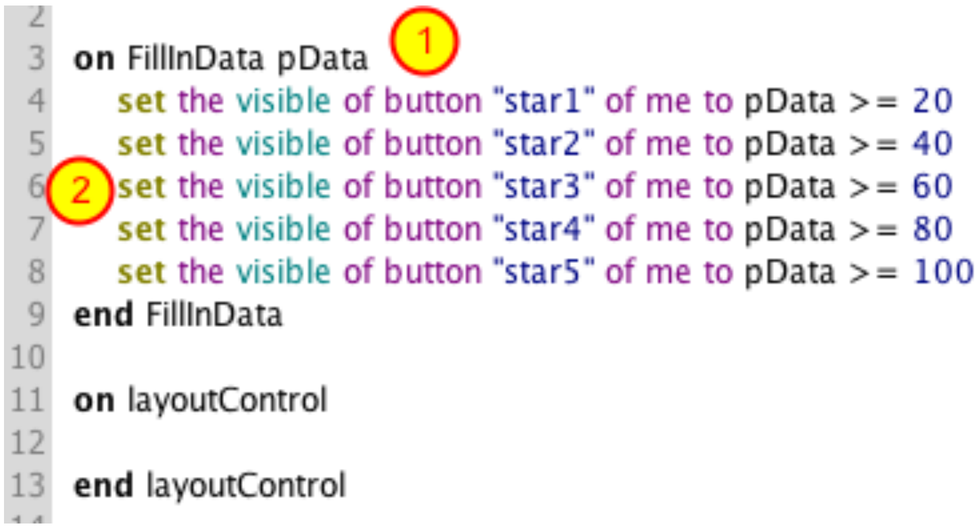
## Supporting Controls



|  | 🔍 | ⬚ | Layer | Control | 📜 |
|---|---|---|---|---|---|
|  | ✓ | ✓ | 1 | Row Template | 0 |
|  | ✓ | ✓ | 2 | Background | 0 |
|  | ✓ | ✓ | 3 | Time | 0 |
|  | ✓ | ✓ | 4 | My Rating | 0 |
|  | ✓ | ✓ | 5 | EventCatcher | 0 |
|  | ✓ | ✓ | 6 | star1 | 0 |
|  | ✓ | ✓ | 7 | star2 | 0 |
|  | ✓ | ✓ | 8 | star3 | 0 |
|  | ✓ | ✓ | 9 | star4 | 0 |
|  | ✓ | ✓ | 10 | star5 | 0 |
|  | ✓ | ✓ | 11 | Genre | 0 |
|  |  | ✓ | 12 | Behavior Script | 24 |
|  | ✓ | ✓ | 13 | Time | 21 |
|  | ✓ | ✓ | 14 | Override Default Column Behavior | 47 |
|  | ✓ | ✓ | 15 | star.png | 0 |
|  | ✓ | ✓ | 16 | My Rating | 25 |
|  | ✓ | ✓ | 17 | _DataGridTemplateInstructions_ | 0 |
|  | ✓ | ✓ | 18 | _DataGridTemplateInstructions_ | 0 |

The controls I created ("Time" field, "Genre" button and "My Rating" group) each have behaviors associated with them. I've stored the buttons containing these behaviors, as well as the "star.png" image on the same card as the "Row Template" group.

## Behavior Example

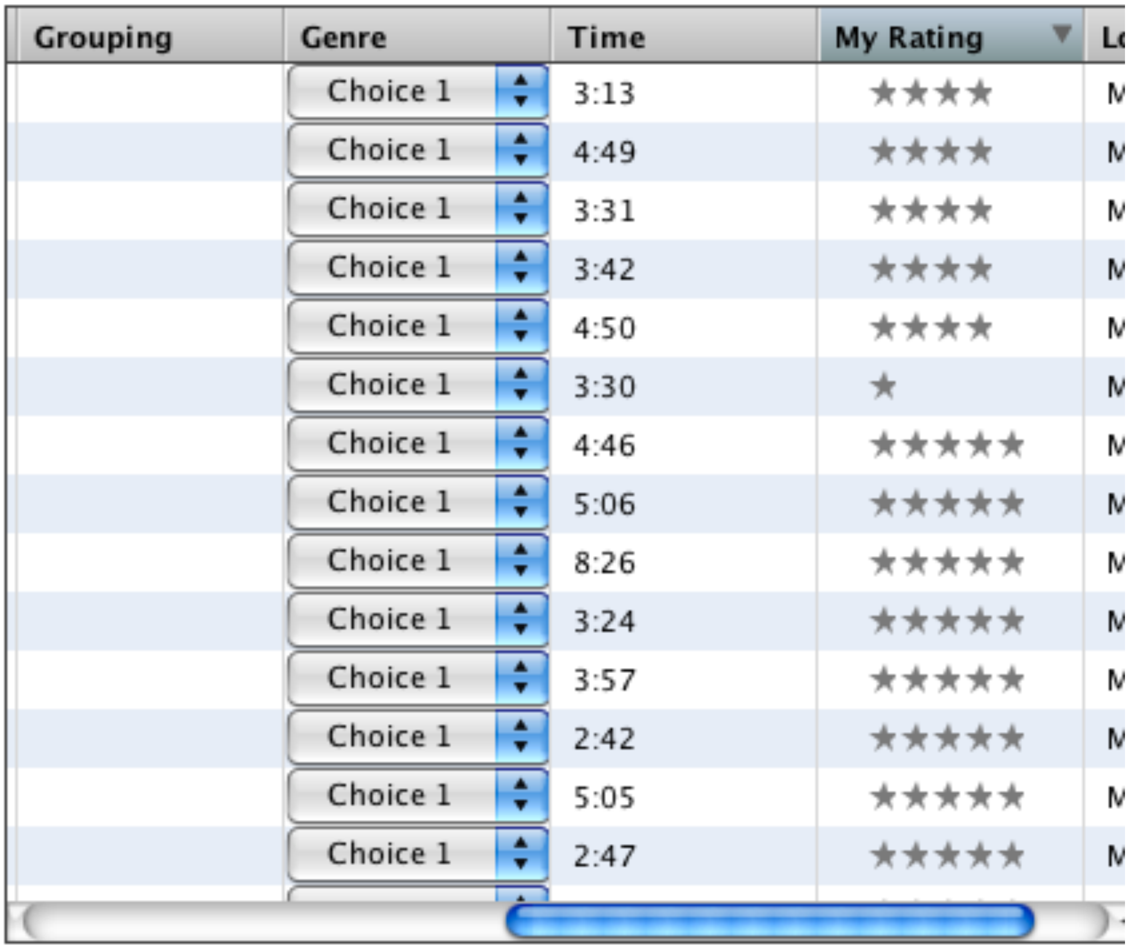


```
on FillInData pData
   set the visible of button "star1" of me to pData >= 20
   set the visible of button "star2" of me to pData >= 40
   set the visible of button "star3" of me to pData >= 60
   set the visible of button "star4" of me to pData >= 80
   set the visible of button "star5" of me to pData >= 100
end FillInData

on layoutControl

end layoutControl
```

Here is what the **FillInData** handler looks like for the "My Rating" column. pData (1) contains the value of the "My Rating" column for the row being displayed. Note how this parameter differs than the parameter sent to **FillInData** for a **row template.** For a **row template** the parameter passed in is an array.

We then use that value to determine how many stars to display (2).

**Result**



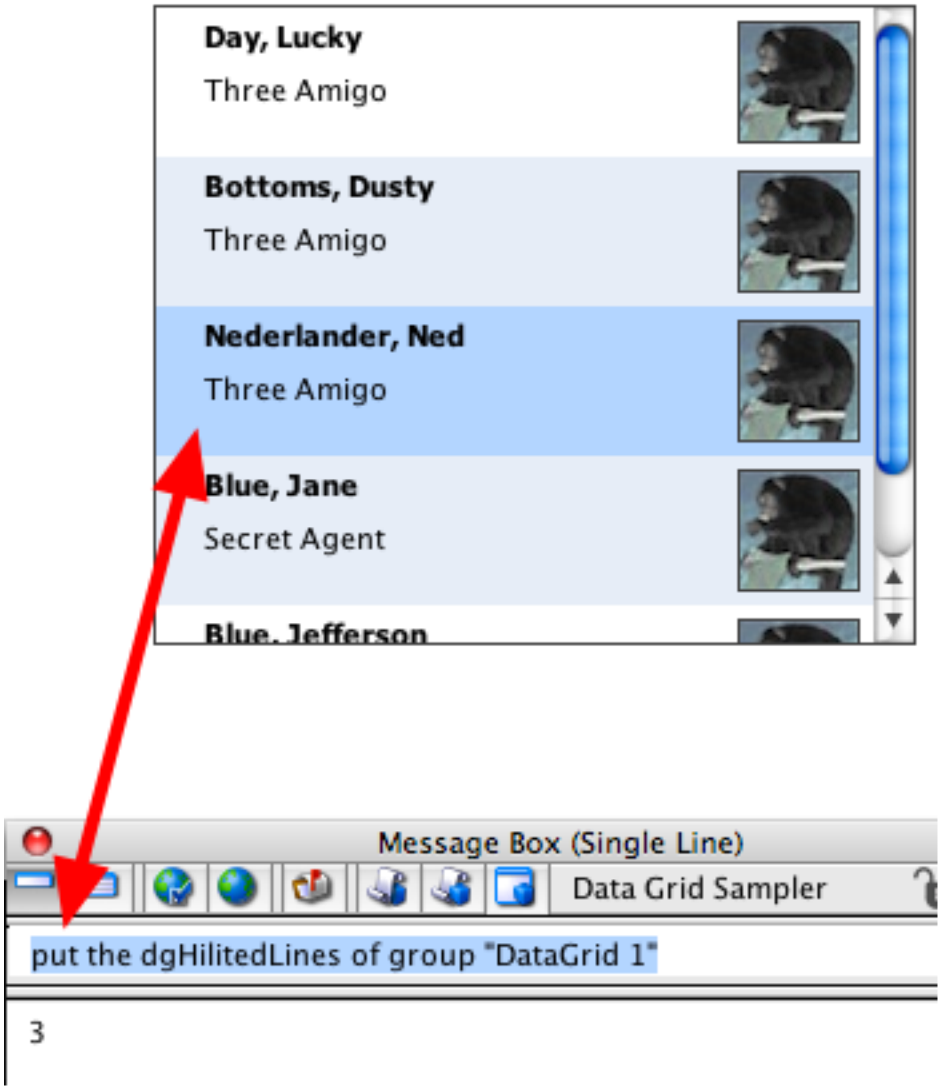
| Grouping | Genre | Time | My Rating | L |
|---|---|---|---|---|
| | Choice 1 | 3:13 | ★★★★ | M |
| | Choice 1 | 4:49 | ★★★★ | M |
| | Choice 1 | 3:31 | ★★★★ | M |
| | Choice 1 | 3:42 | ★★★★ | M |
| | Choice 1 | 4:50 | ★★★★ | M |
| | Choice 1 | 3:30 | ★ | M |
| | Choice 1 | 4:46 | ★★★★★ | M |
| | Choice 1 | 5:06 | ★★★★★ | M |
| | Choice 1 | 8:26 | ★★★★★ | M |
| | Choice 1 | 3:24 | ★★★★★ | M |
| | Choice 1 | 3:57 | ★★★★★ | M |
| | Choice 1 | 2:42 | ★★★★★ | M |
| | Choice 1 | 5:05 | ★★★★★ | M |
| | Choice 1 | 2:47 | ★★★★★ | M |

Here is what the table looks like now that I've defined some custom column templates. The table found controls named "Genre", "Time" and "My Rating" in the record template group so those were used to render the data for those three columns.

# Working With Data Grids (Forms & Tables)

## How Do I Determine The Selected Line?

### the dgHilitedLines Property

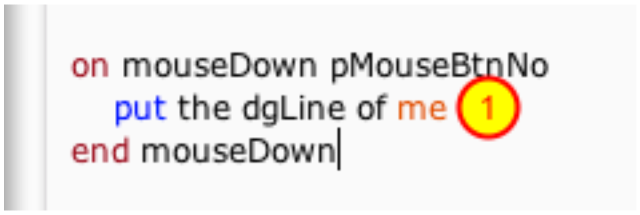


```
put the dgHilitedLines of group "DataGrid 1"
```

```
3
```

You can get the selected line or lines of a data grid using the dgHilitedLines property.

### Determining the Line In A Row's Behavior



```
on mouseDown pMouseBtnNo
   put the dgLine of me
end mouseDown
```

If you need to determine the line number of a row's behavior script you can access the dgLine property of the row template. The data grid automatically assigns the dgLine and the dgIndex property when displaying the row.

Note that the use of 'of me' (1) can be used because the mouseDown handler is in the row template group behavior script (which is as if the script were in the row template group itself). If you were to put the mouseDown handler in the script of a button that was located in the row template group then you would have to use 'of the dgControl of me' instead.

## How Do I Get Data Associated With a Row or Column?

This lesson will show you how to get the array of data associated with a row in a Data Grid form as well as how to get the data associated with a row's column in a Data Grid table.

To get the array of data from a Data Grid you can use one of the following properties:

- the **dgDataOfIndex**
- the **dgDataOfLine**

Both properties return the array associated with a row in the Data Grid, they just allow you to target the row differently. dgDataOfLine allows you to target the row in the order it appears visually in the Data Grid. dgDataOfIndex allows you to target a row based on the internal index number that the Data Grid uses to identify the row. This value does not change, even if you change the sort order for the Data Grid.

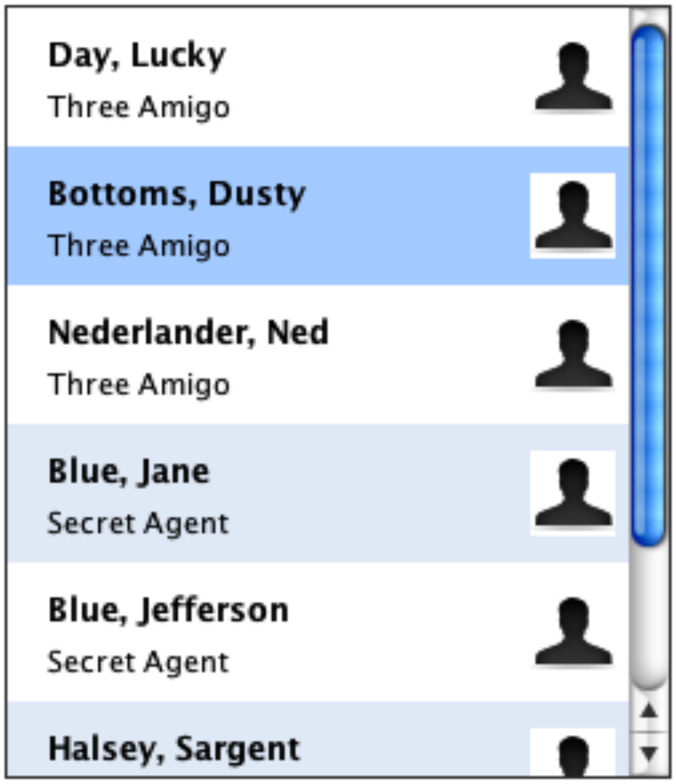
If you just need a particular column from a row then you can use one of the following functions:

- **GetDataOfIndex()**
- **GetDataOfLine()**

Both of these functions return the value associated with a particular key, or column, for a row.

Let's look at some examples of how to use these properties and functions.

## Getting Data Associated with the Selected Row



If you want to get the data associated with the selected row use the **dgHilitedLines** property. The value of that property can be used in conjunction with the dgDataOfLine property. Here is an example of a script that could be placed in a button. The Data Grid group script is NOT in the message path for this example:

```
on mouseUp pBtnNum
   if pBtnNum is 1 then
      put the dgHilitedLines of group "DataGrid 1" into theLine
      put the dgDataOfLine[theLine] of group "DataGrid 1" into theDataA
      ## theDataA is now an array variable.
      ## In the case of the Data Grid pictured above the keys of the array are id, FirstName,
LastName and Title.

      ## This answer dialog will display: Dusty Bottoms
      answer theDataA["FirstName"] && theDataA["LastName"]
   end if
end mouseUp
```

Now let's assume that you only want to get the "id" value for the selected row without fetching the entire array. You can use **GetDataOfLine** to accomplish this. You need to keep in mind that GetDataOfLine (and GetDataOfIndex) is a function defined for a Data Grid. That means that the Data Grid group MUST be in the message path. If the Data Grid group is not in the message path you would need to use the call command.

To illustrate how to use GetDataOfLine let's look at how you could define your own custom property for a Data Grid. The following code defines a uSelectedID custom property that returns the id of the selected row. This code would be placed in the Data Grid group script.

```
getProp uSelectedID
   put the dgHilitedLines of me into theLine
   return GetDataOfLine(theLine, "id")
end uSelectedID
```

You can now access this custom property from any script in your application:

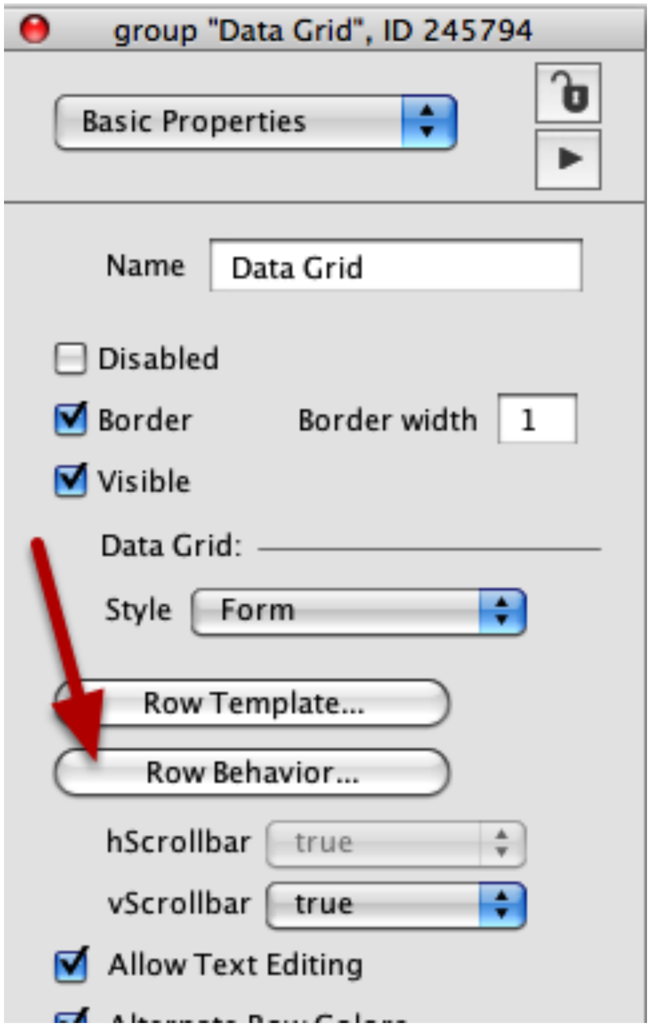
```
put the uSelectedID of group "DataGrid" into theSelectedID
```

## Getting Data When The Selection Changes

When the user makes a new selection in a Data Grid the selectionChanged message is sent to the Data Grid. The following selectionChanged message could appear in the script of your Data Grid group:

```
-- example that would be placed in the data grid script.
on selectionChanged pHilitedIndex, pPrevHilitedIndex
   put the dgDataOfIndex [ pHilitedIndex ] of me into theDataA

   ## Now call handler in card script that loads info for selected person.
   uiViewRecordOfID theDataA["id"]
end selectionChanged
```

## Getting Data In A Row Behavior



If you want to get the data in a row behavior script you can use the dgIndex or dgLine custom properties of the row control. Here is an example that could be used in a Data Grid form behavior script:

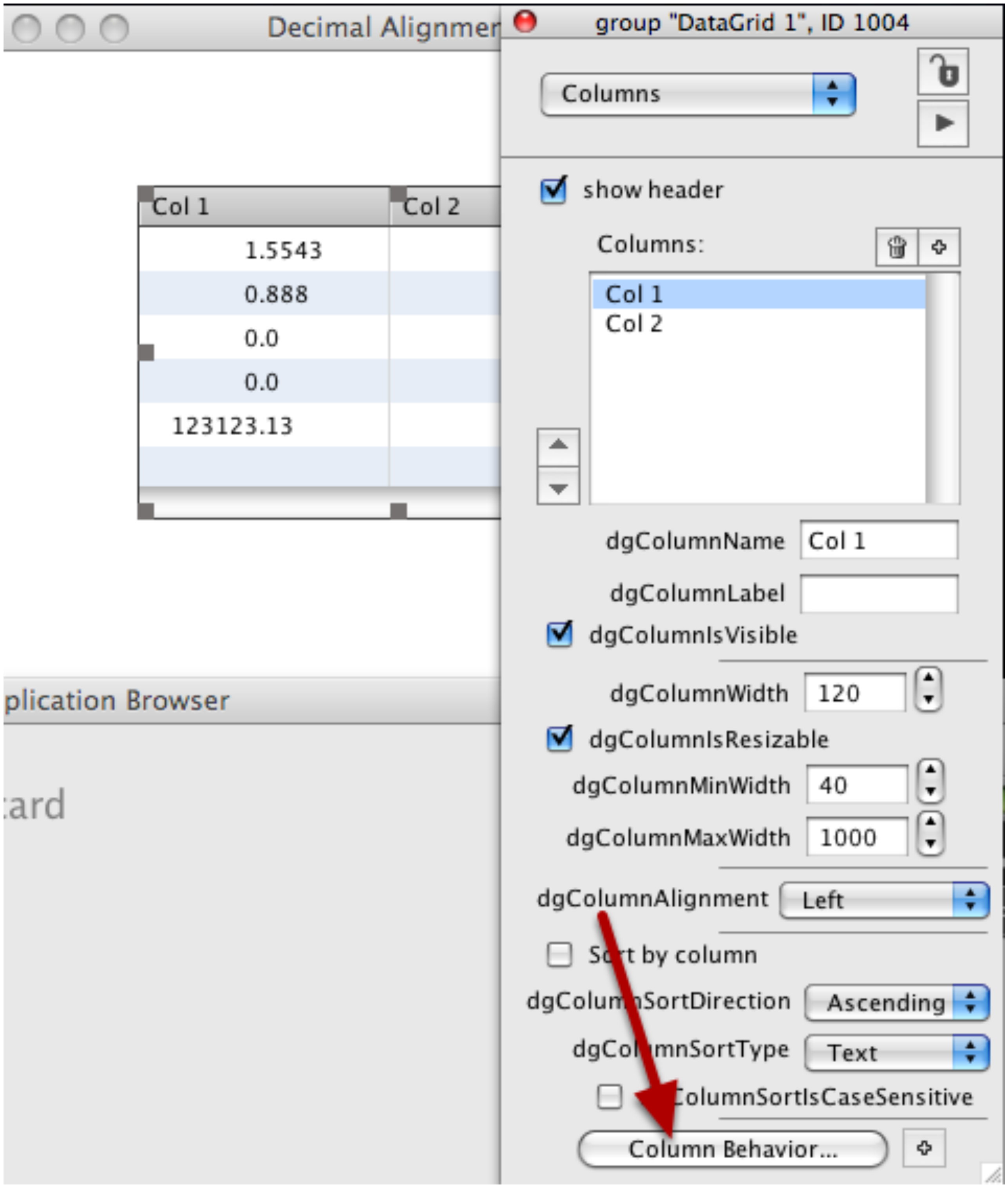
**==========**
***Data Grid Form Example: Row Behavior***
**==========**

```
on mouseUp pMouseBtnNum
   if pMouseBtnNum is 1 then
      ## the dgIndex is a custom property of this row.
      ## the dgControl is a custom property of the data grid itself.
      put the dgDataOfIndex[ the dgIndex of me] of the dgControl of me into theDataA
      uiViewRecordOfID theDataA["id"]
   end if
end mouseUp
```

If you wanted to move this same mouseUp code into the Data Grid script itself then you would change the 'me' references to 'target' and add a check to ensure that the user clicked on a row. This example also uses GetDataOfIndex rather than dgDataOfIndex to show an alternative.

```
==========
Data Grid Form Example: Data Grid Script
==========
on mouseUp pMouseBtnNum
   if pMouseBtnNum is 1 then
      ## the dgIndex is a custom property of the row.
      put the dgIndex of the target into theIndexThatWasClickedOn
      if theIndexThatWasClickedOn is not empty then
         put GetDataOfIndex(theIndexThatWasClickedOn, "id") into theID
         uiViewRecordOfID theID
      end if
   end if
end mouseUp
```

## Getting Data In A Column Behavior


Decimal Alignment

group "DataGrid 1", ID 1004

Columns

| Col 1 | Col 2 |
|---|---|
| 1.5543 | |
| 0.888 | |
| 0.0 | |
| 0.0 | |
| 123123.13 | |

[✓] show header

Columns:

Col 1

Col 2

dgColumnName: Col 1

dgColumnLabel:

[✓] dgColumnIsVisible

dgColumnWidth: 120

[✓] dgColumnIsResizable

dgColumnMinWidth: 40

dgColumnMaxWidth: 1000

dgColumnAlignment: Left

[ ] Sort by column

dgColumnSortDirection: Ascending

dgColumnSortType: Text

[ ] dgColumnSortIsCaseSensitive

If you want to get the data in a column behavior script you can use the dgIndex or dgLine and dgColumn custom properties of the column control. Here is an example that could be used in a Data Grid column behavior script:

==========

***Data Grid Table Example: Column Behavior***

```
==========
on mouseUp pMouseBtnNum
   if pMouseBtnNum is 1 then
      ## the dgIndex is a custom property of the row.
      ## the dgColumn is a custom property of the column.
      put GetDataOfIndex(the dgIndex of me, the dgColumn of me) into theColumnValue

      ## Do something with column value..
   end if
end mouseUp
```

If you wanted to move this same mouseUp code into the Data Grid script itself then you would change the 'me' references to 'target' and add a check to ensure that the user clicked on a column:

```
==========
```

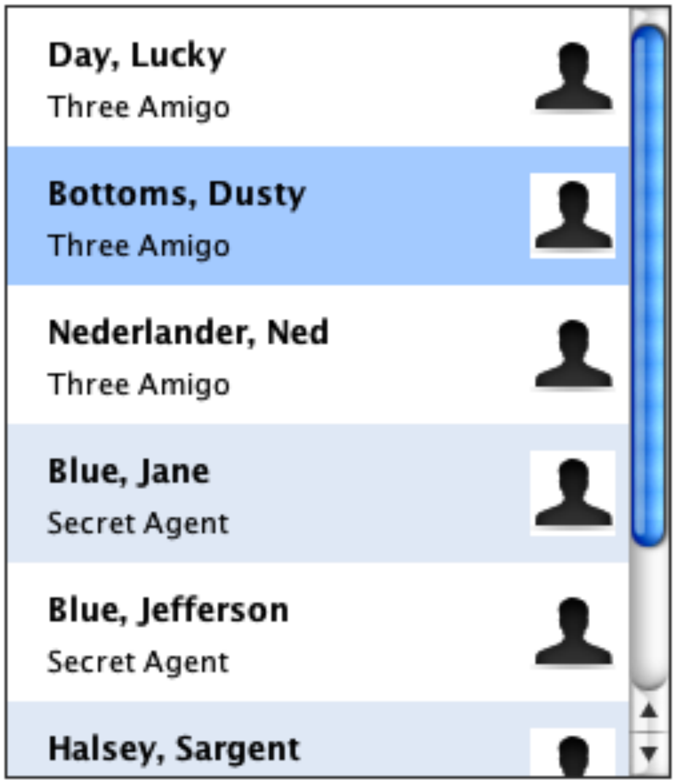
***Data Grid Table Example: Data Grid Script***

```
==========
on mouseUp pMouseBtnNum
   if pMouseBtnNum is 1 then
      ## the dgIndex is a custom property of the row.
      ## the dgColumn is a custom property of the column.
      put the dgColumn of the target into theColumn
      if theColumn is not empty then ## User clicked on a column
         put GetDataOfIndex(the dgIndex of the target, theColumn) into theColumnValue

         ## Do something with column value..
      end if
   end if
end mouseUp
```

## How Do I Add A Row Of Data To An Existing Data Grid?

Sometimes you may want to add a row of data to a Data Grid without having to set the dgData or dgText property. You can add a single row of data by calling the **AddData** or **AddLine** commands of a Data Grid.

### Using AddData

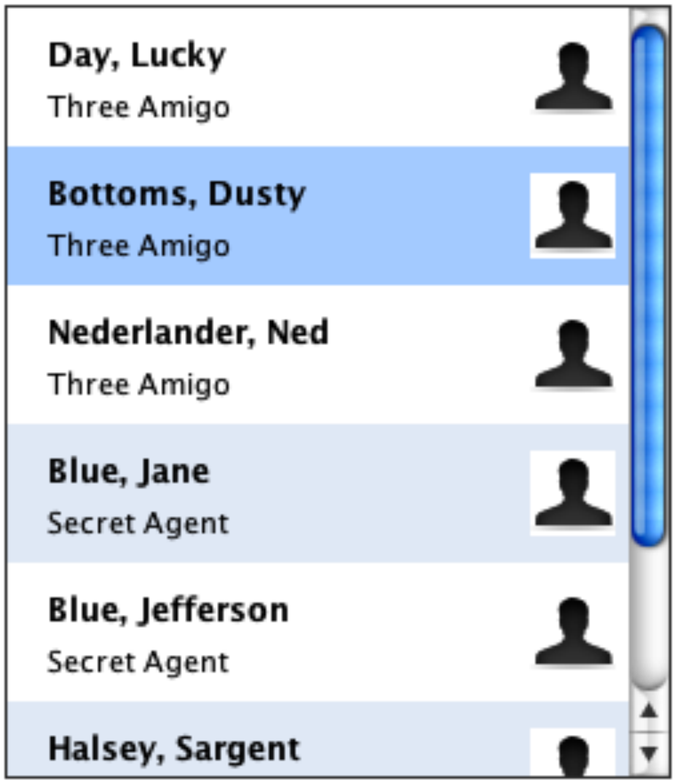


Day, Lucky
Three Amigo
Bottoms, Dusty
Three Amigo
Nederlander, Ned
Three Amigo
Blue, Jane
Secret Agent
Blue, Jefferson
Secret Agent
Halsey, Sargent

To use **AddData** you create an array containing the values for the new row. Here is an example of how to add a new row and have it appear as line 1 in a Data Grid. This example script resides outside of the Data Grid group script so AddData is not in the message path. This is why the dispatch command is used.

```
put "First Name" into theDataA["FirstName"]
put "Last Name" into theDataA["LastName"]
put "Title" into theDataA["Title"]

put  1 into theLineNo

dispatch "AddData" to group "DataGrid" with theDataA, theLineNo
put the result into theNewIndex -- integer if successful, error string otherwise
```

## Using AddLine



To use **AddLine** you create a tab delimited string of text containing the values for the new row. You also need to tell the Data Grid the names of the columns that the data should map to. Here is an example of how to add a new row and have it appear as the last line in a Data Grid. This example script resides inside the Data Grid group script so AddLine is in the message path.

```
put "First Name" & tab & "Last Name" & tab & "Title" into theRowData
put "FirstName" & cr & "LastName" & cr & "Title" into theDataColumns

put the dgNumberOfLines of me + 1 into theLineNo

AddLine theRowData, theDataColumns, theLineNo
```

## Scrolling Data Into View And Getting The Data Control

After you add the data to the Data Grid you may want to scroll the new row into view. You can call **ScrollIndexIntoView** or **ScrollLineIntoView** to do this.

```
ScrollIndexIntoView theNewIndex
```

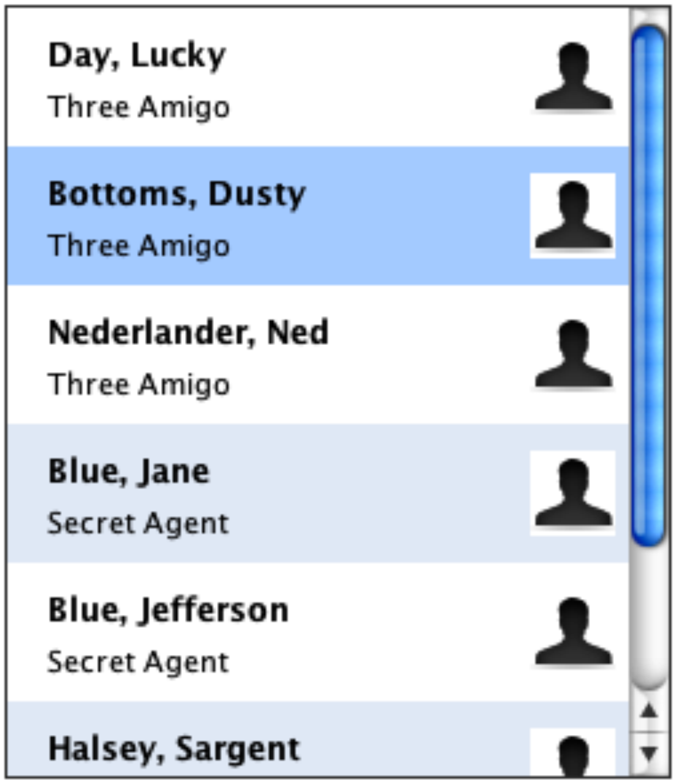
or

```
ScrollLineIntoView theLineNo
```

## How Do I Update Data In a Row?

This lesson will show you how to update data in a row.

### Updating A Row's Data And Refresh Data Grid Automatically



You can update data in a row by setting the **dgDataOfIndex** or **dgDataOfLine** properties. You can set either property to an array containing the values for the row.

```
put "New First Name" into theDataA["FirstName"]
put "New Last Name" into theDataA["LastName"]
put "New Title" into theDataA["Title"]

set the dgDataOfIndex[ the dgHilitedIndex of group "DataGrid" ] of group "DataGrid" to theDataA
```

The data grid will now refresh with the new data you assigned to the highlighted index.

**Note:** dgDataOfLine does not automatically refresh the data grid in the version that ships with Revolution 3.5 gm-2 (1.0.0 build 8). Please use dgDataOfIndex unless you have a later version.

## Updating a Row's Data Without Refreshing the Data Grid

If you want to update the data in a row without automatically refreshing the data grid then you can use *SetDataOfIndex*. You can use the *RefreshList* command to redraw the data grid in this case.

Example that sets all values at once:

```
put "New First Name" into theDataA["FirstName"]
put "New Last Name" into theDataA["LastName"]
put "New Title" into theDataA["Title"]

dispatch "SetDataOfIndex" to group "DataGrid" \
        with the dgHilitedIndex of group "DataGrid", empty, theDataA

dispatch "RefreshIndex" to group "DataGrid" with the dgHilitedIndex of group "DataGrid"
```

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

```
dispatch "SetDataOfIndex" to group "DataGrid" \
        with the dgHilitedIndex of group "DataGrid", "FirstName", "New First Name"

dispatch "SetDataOfIndex" to group "DataGrid" \
        with the dgHilitedIndex of group "DataGrid", "LastName", "New Last Name"

dispatch "SetDataOfIndex" to group "DataGrid" \
        with the dgHilitedIndex of group "DataGrid", "Title", "New Title"

dispatch "RefreshIndex" to group "DataGrid" with the dgHilitedIndex of group "DataGrid"
```

## How Do I Clear Data From a Data Grid?

Clearing the data out of a Data Grid is as easy as setting the dgData (or dgText) of the Data Grid to empty.

```
set the dgData of group "DataGrid" to empty
```

## How Do I Add a mouseDown Event To The Data Grid Without Breaking It?

This lesson will show you how you to write your own mouseDown event in a Data Grid without breaking the default Data Grid behavior. You need to know how to do this when showing a contextual menu or if during the mouseDown event you want to use the data in the row the user clicked on.

When you add a mouseDown handler to a Data Grid you are intercepting a message that the Data Grid normally handles. Doing so changes the behavior of the Data Grid and you need to take that into account when coding your mouseDown handler.

### What Doesn't Work

If you were to place the following code in your Data Grid script you would not get the result you were expecting.  The reason is that the line that was clicked on would not be selected until after your popup menu was displayed. Why? Because the Data Grid behavior script processes mouseDown AFTER the mouseDown handler you define in the Data Grid script itself.

```
on mouseDown pMouseBtnNum
   if pMouseBtnNum is 3 then
      ## Oops! Line that user clicked on has not been selected since
      ## Data Grid has not processed mouseDown yet. Contextual
      ## menu won't target proper line.
      popup button "MyContextualMenu"
   end if
   pass mouseDown
end mouseDown
```

### What Does Work: dgMouseDown

In order to work around this the Data Grid wraps all mouseDown functionality in a handler named **dgMouseDown**. You can call this handler in your code in order to get the expected results.

The following examples show how you can define a mouseDown handler in a Data Grid script.  In the examples the line the user clicks on will be selected before the custom mouseDown code executes. Note that I don't pass mouseDown after calling dgMouseDown. This would only repeat the call to dgMouseDown which I don't want to do.

**==========**
***Example: Contextual Menu***
**==========**

```
on mouseDown pMouseBtnNum
   ## Let Data Grid process mouseDown and select row that was clicked on
```

```
    dgMouseDown pMouseBtnNum

    ## Now contextual menu will act on the proper line.
    if pMouseBtnNum is 3 then
        popup button "MyContextualMenu"
    end if

    ## Don't pass mouseDown
end mouseDown
```

**==========**

***Example: Getting Value From Cell Clicked On (Data Grid table)***

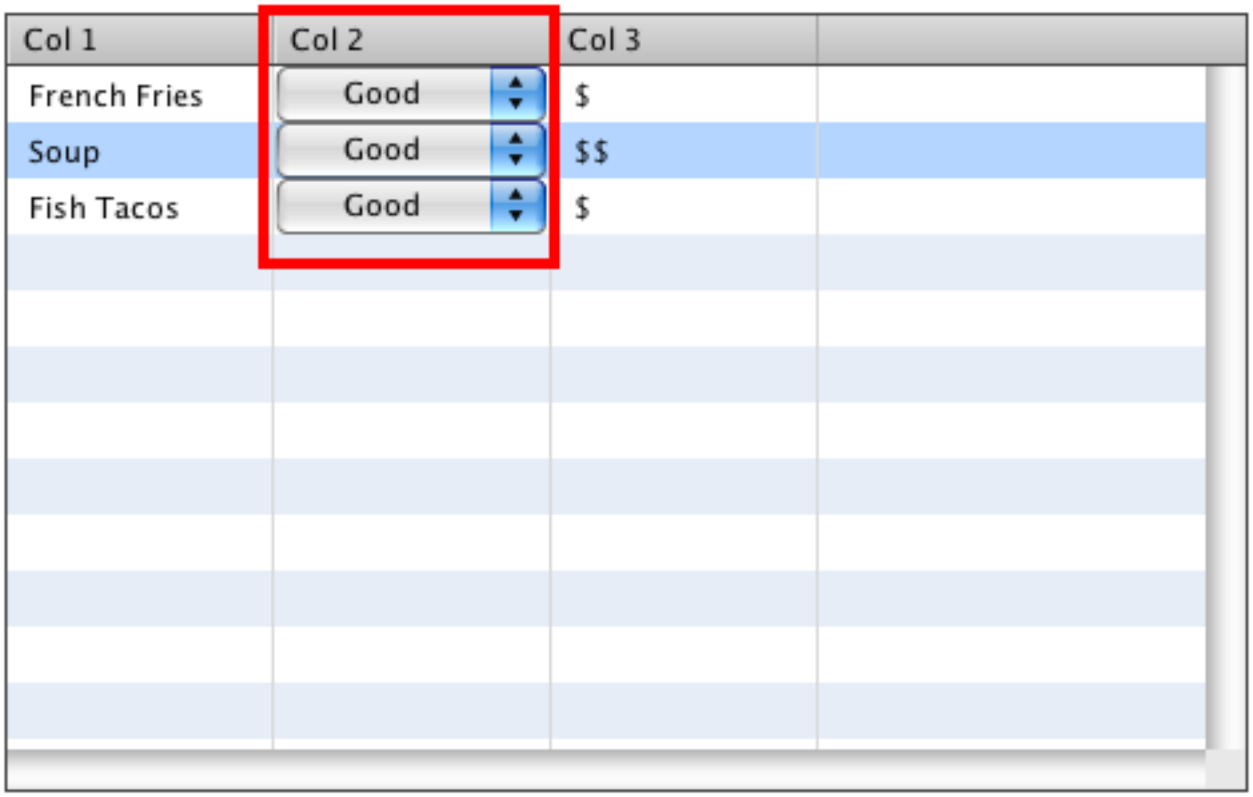
**==========**

```
on mouseDown pMouseBtnNum
    ## Let Data Grid process mouseDown and select row that was clicked on
    dgMouseDown pMouseBtnNum

    ## Get value of column clicked on. The column name can be accessed in the
    ## dgColumn custom property of the column control (the target).
    put GetDataOfIndex(the dgHilitedIndexes of me, the dgColumn of the target) into theColumnValue

    ## Don't pass mouseDown
end mouseDown
```

## How Can I Store An Option Menu Value When The User Makes a Selection?

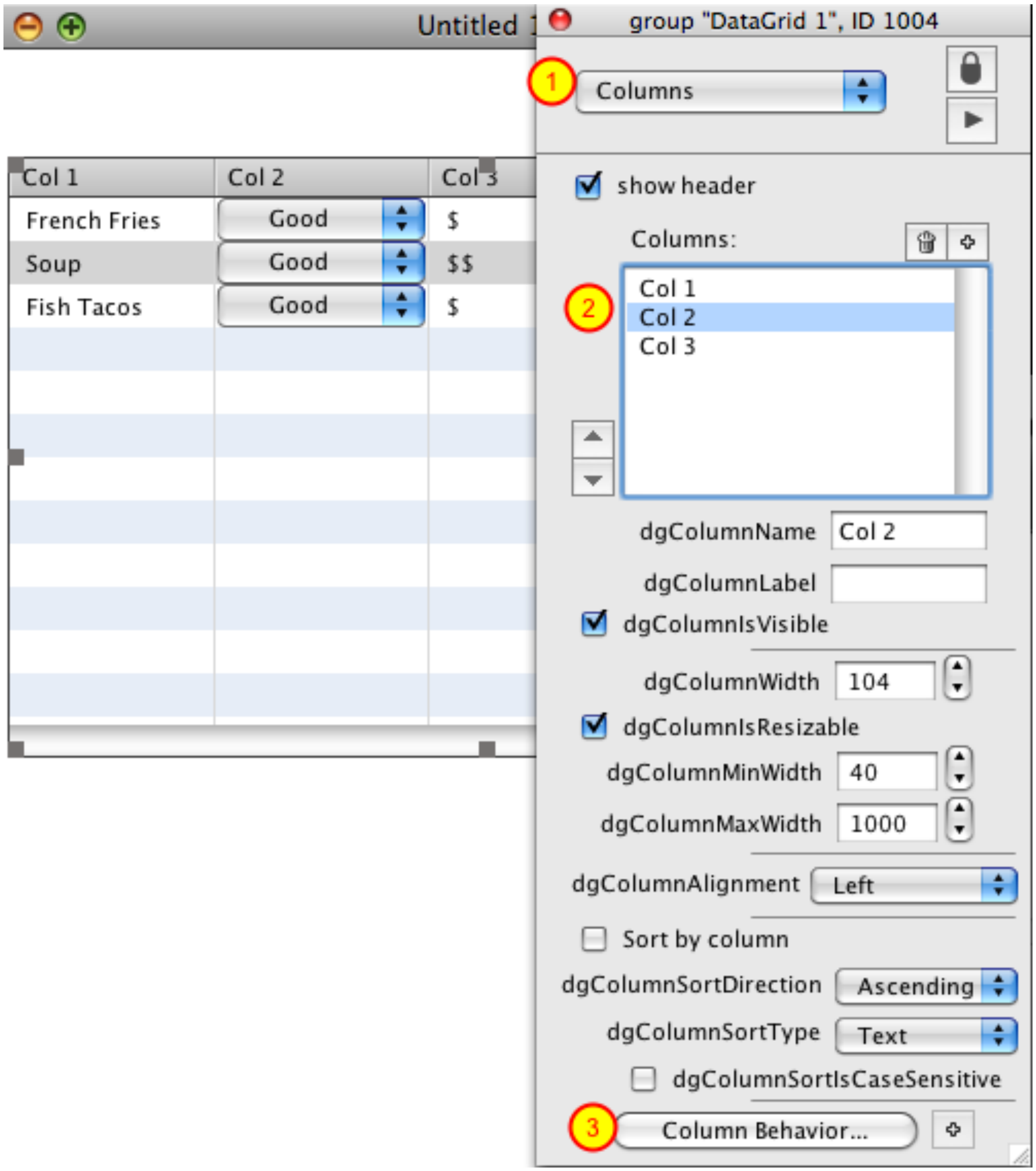
This lesson will demonstrate how to update the data associated with a row in a data grid when the user makes a selection from an option menu.

### Example Data Grid



Here is the data grid we will be working with. **Col 2** has been customized with an option menu. What we are going to do is update the data associated with a row to reflect the selection the user makes in the option menu.

## Edit Column Behavior



| Col 1 | Col 2 | Col 3 |
|---|---|---|
| French Fries | Good | $ |
| Soup | Good | $$ |
| Fish Tacos | Good | $ |

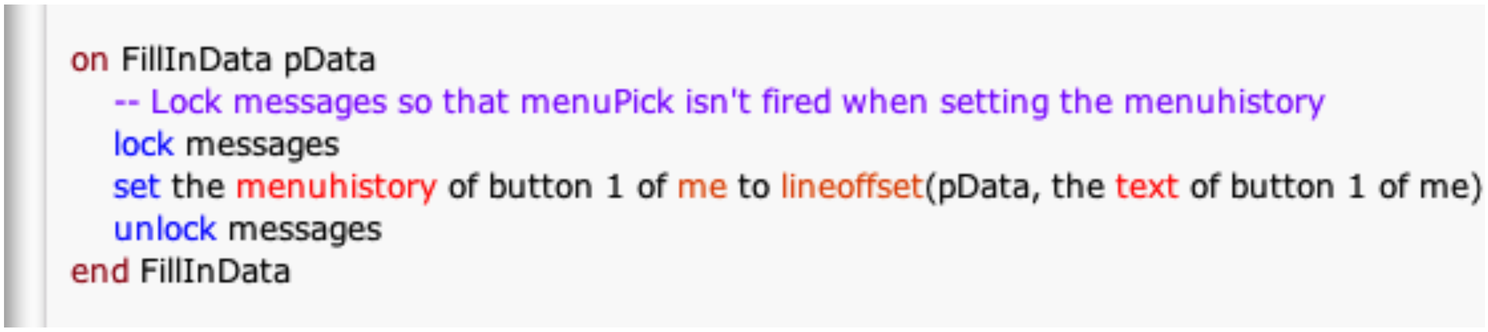
show header: ✓

Columns:
- Col 1
- Col 2
- Col 3

| | |
|---|---|
| dgColumnName | Col 2 |
| dgColumnLabel | |
| dgColumnIsVisible | ✓ |
| dgColumnWidth | 104 |
| dgColumnIsResizable | ✓ |
| dgColumnMinWidth | 40 |
| dgColumnMaxWidth | 1000 |
| dgColumnAlignment | Left |
| Sort by column | |
| dgColumnSortDirection | Ascending |
| dgColumnSortType | Text |
| dgColumnSortIsCaseSensitive | |

We need to customize the column behavior. From the **Columns** pane (1) in the Property Inspector select **Col 2** and then click the **Column Behavior** button (3).

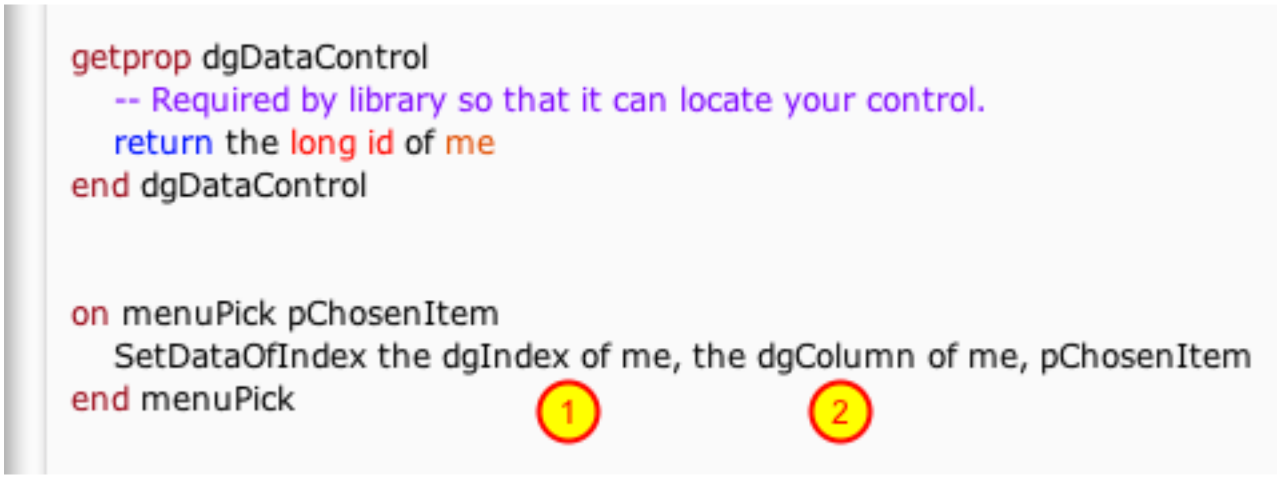
## FillInData



```
on FillInData pData
   -- Lock messages so that menuPick isn't fired when setting the menuhistory
   lock messages
   set the menuhistory of button 1 of me to lineoffset(pData, the text of button 1 of me)
   unlock messages
end FillInData
```

In the **FillInData** handler we are setting the menuhistory based on the value of pData that is passed in.

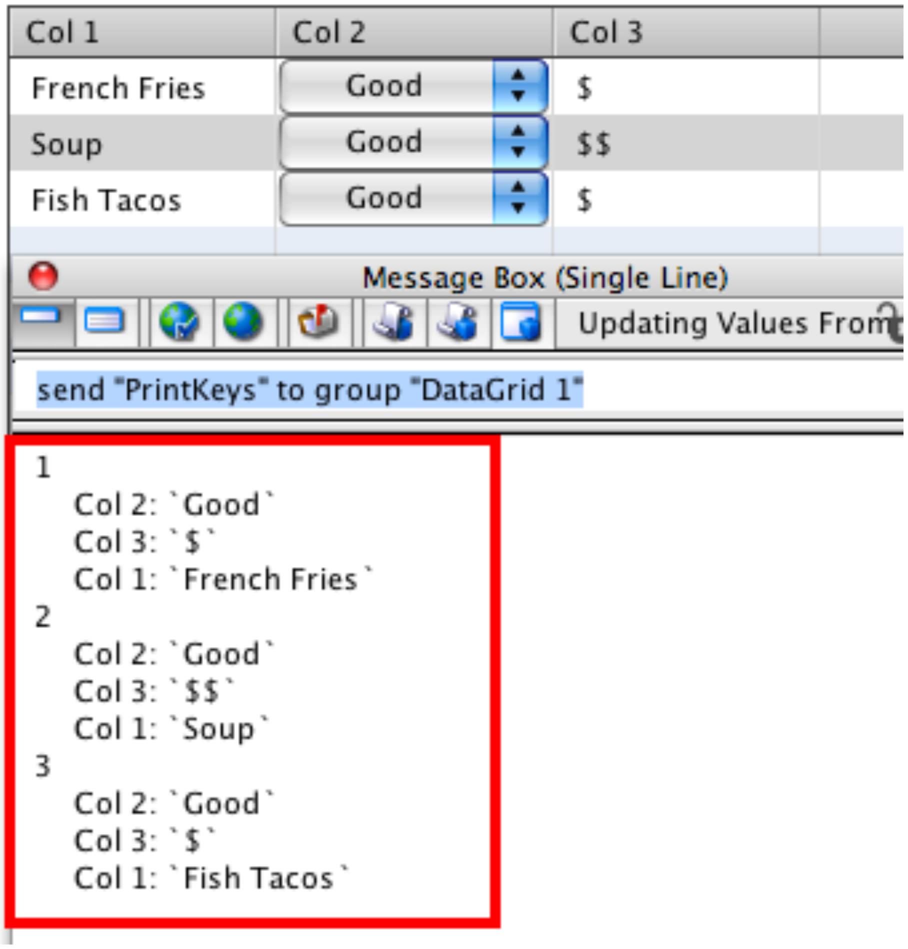
## menuPick



```
getprop dgDataControl
   -- Required by library so that it can locate your control.
   return the long id of me
end dgDataControl


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

We can add a a menuPick handler to the column behavior script in order to accomplish our goal. When the user makes a selection from the option menu the **menuPick** message is sent by the engine. We can then use the **SetDataOfIndex** command to update the value for the column. The parameters for SetDataOfIndex are the index, the column name and the new value. Since the above script is in a column behavior we can use the dgIndex of me (1) for the index and the dgColumn of me (2) for the column name.

## The Behavior In Action

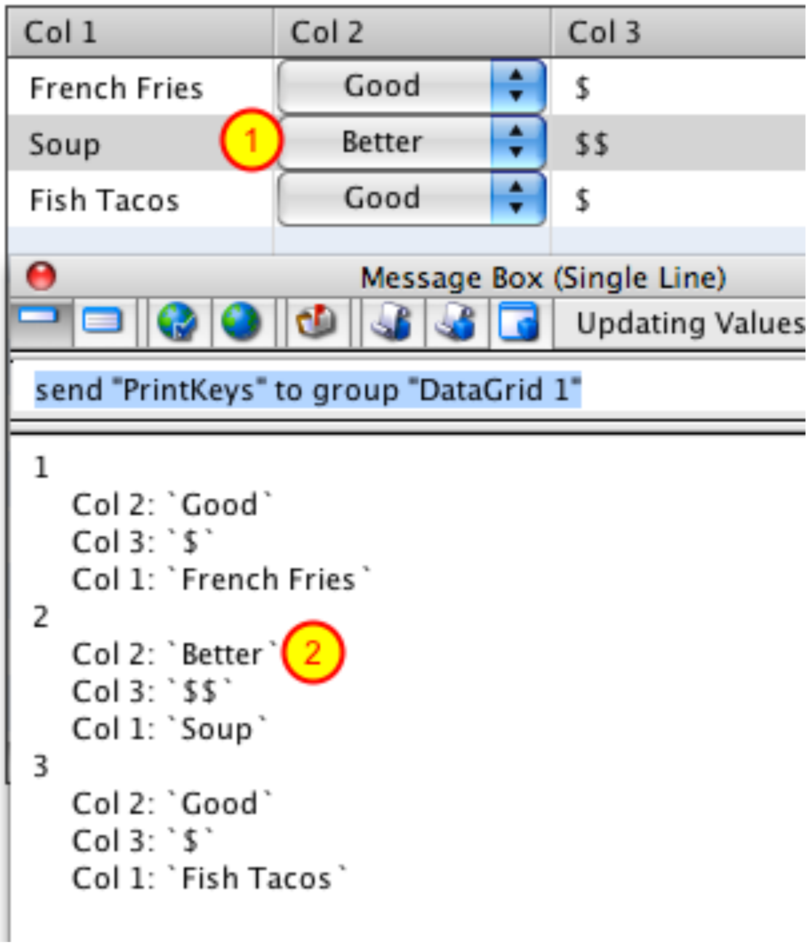


| Col 1 | Col 2 | Col 3 |
|---|---|---|
| French Fries | Good | $ |
| Soup | Good | $$ |
| Fish Tacos | Good | $ |

```
send "PrintKeys" to group "DataGrid 1"
```

```
1
    Col 2: `Good`
    Col 3: `$`
    Col 1: `French Fries`
2
    Col 2: `Good`
    Col 3: `$$`
    Col 1: `Soup`
3
    Col 2: `Good`
    Col 3: `$`
    Col 1: `Fish Tacos`
```

Here is what the data grid's internal array looks like before making a menu selection.



```
send "PrintKeys" to group "DataGrid 1"
```

```
1
   Col 2: `Good`
   Col 3: `$`
   Col 1: `French Fries`
2
   Col 2: `Better`
   Col 3: `$$`
   Col 1: `Soup`
3
   Col 2: `Good`
   Col 3: `$`
   Col 1: `Fish Tacos`
```

After making a selection for row 2 (1) we can see that the internal value was updated (2).

## How Do I Refresh a Data Grid After Making Changes to a Template Through Script?

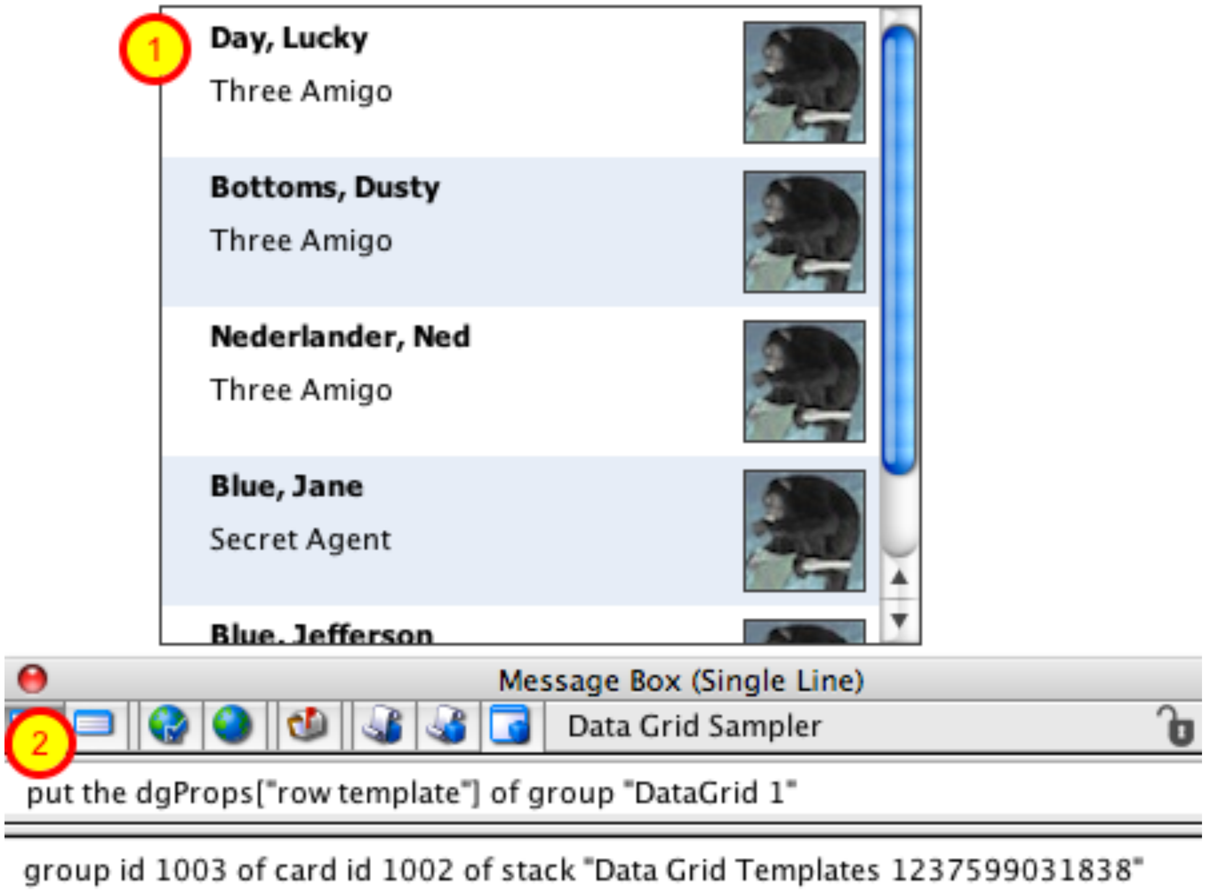
The command **ResetList** redraws a data grid after having copied in fresh copies of any templates. Here is an example of how you might use it.

```
## Get reference to group that serves as row template for data grid
put the dgProps["row template"] of group "DataGrid" into theRowTemplate

## For tables get reference to custom template for column
put the long id of group "My Column" of theRowTemplate into theColTemplate

## Make any updates to template
set the text of button 1 of theColTemplate to "New Option1" & cr & "New Option 2"

## Refresh the data grid
dispatch "ResetList" to group "DataGrid"
```

## How Do I Use A Template In Multiple Data Grids?

If your application needs to use the same style of data grid in multiple places in your project then you can easily share the same row template between multiple data grids. This lesson will show you how.

### Create First Data Grid

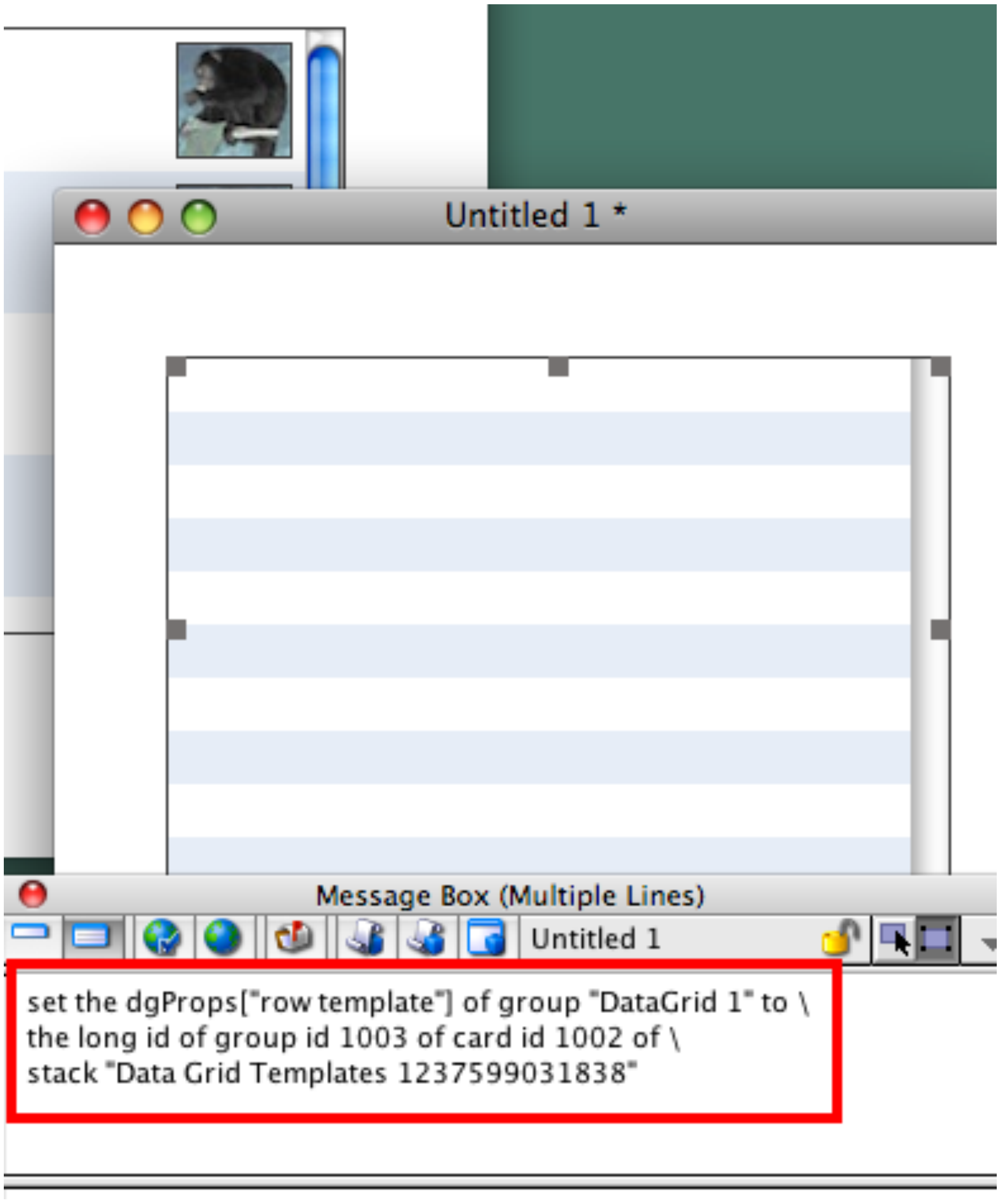


```
put the dgProps["row template"] of group "DataGrid 1"
```

```
group id 1003 of card id 1002 of stack "Data Grid Templates 1237599031838"
```

The first step is to create your first data grid and customize the template (1). The data grid property that determines the **Row Template** that is used to draw the data grid is *dgProps["row template"]*. In the message box you can see that I queried this property in order to get a reference to the data grids row template (2).

## Create Other Data Grids



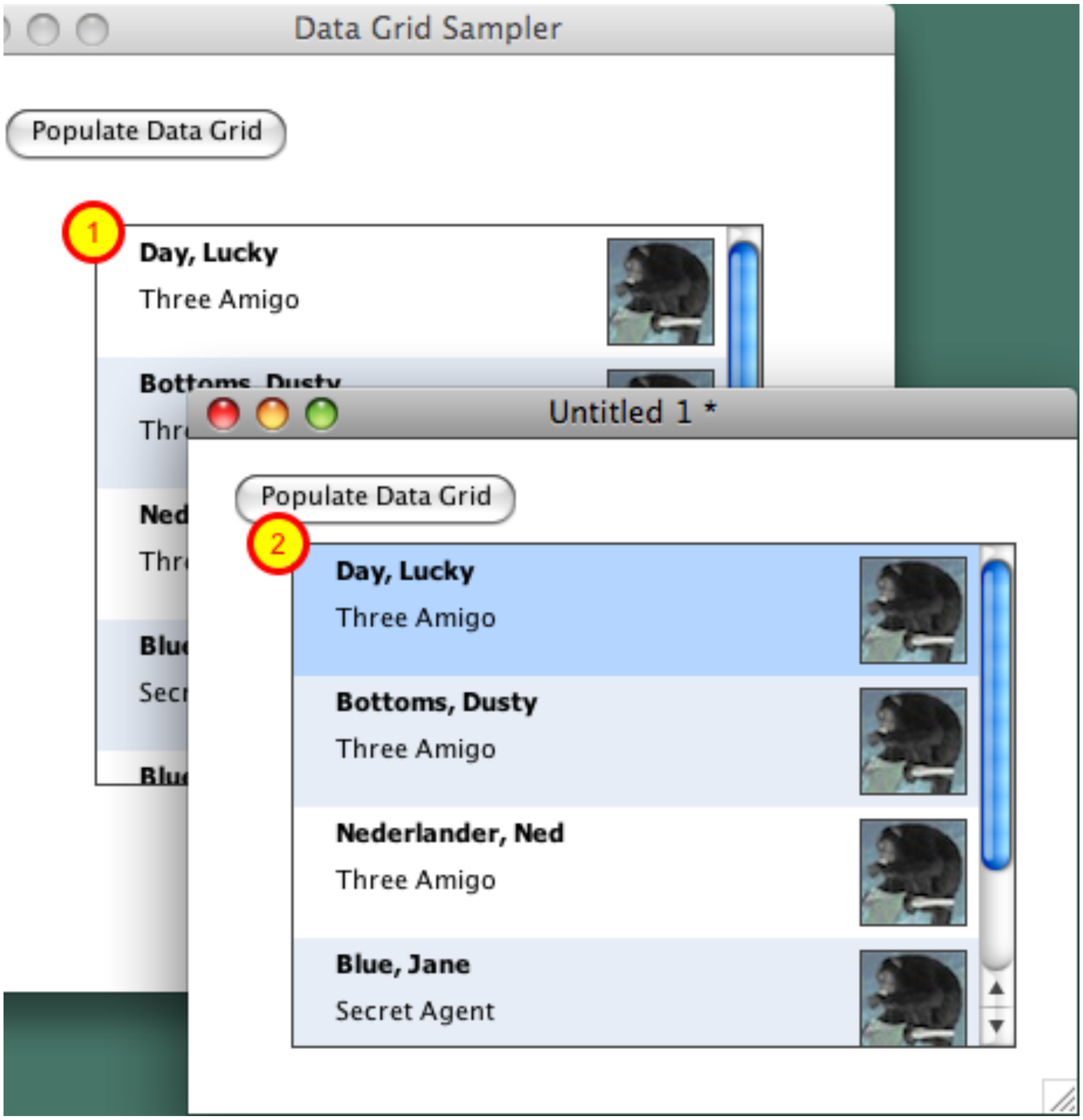
```
set the dgProps["row template"] of group "DataGrid 1" to \
the long id of group id 1003 of card id 1002 of \
stack "Data Grid Templates 1237599031838"
```

Now you can add other data grids to your project. After adding the data grid you can set the *dgProps["row template"]* property to the long id of the first data grid's row template. Alternatively you could also do the following:

set the dgProps["row template"] of group "DataGrid 1" of stack "Untitled 1" to the dgProps["row template"] of group "DataGrid 1" of stack "my program"

**Note:** The Revolution IDE creates a card in the "Data Grid Template XXX" stack with a row template every time you drag a data grid on to a card. You may want to delete this card if you aren't going to be using the template.

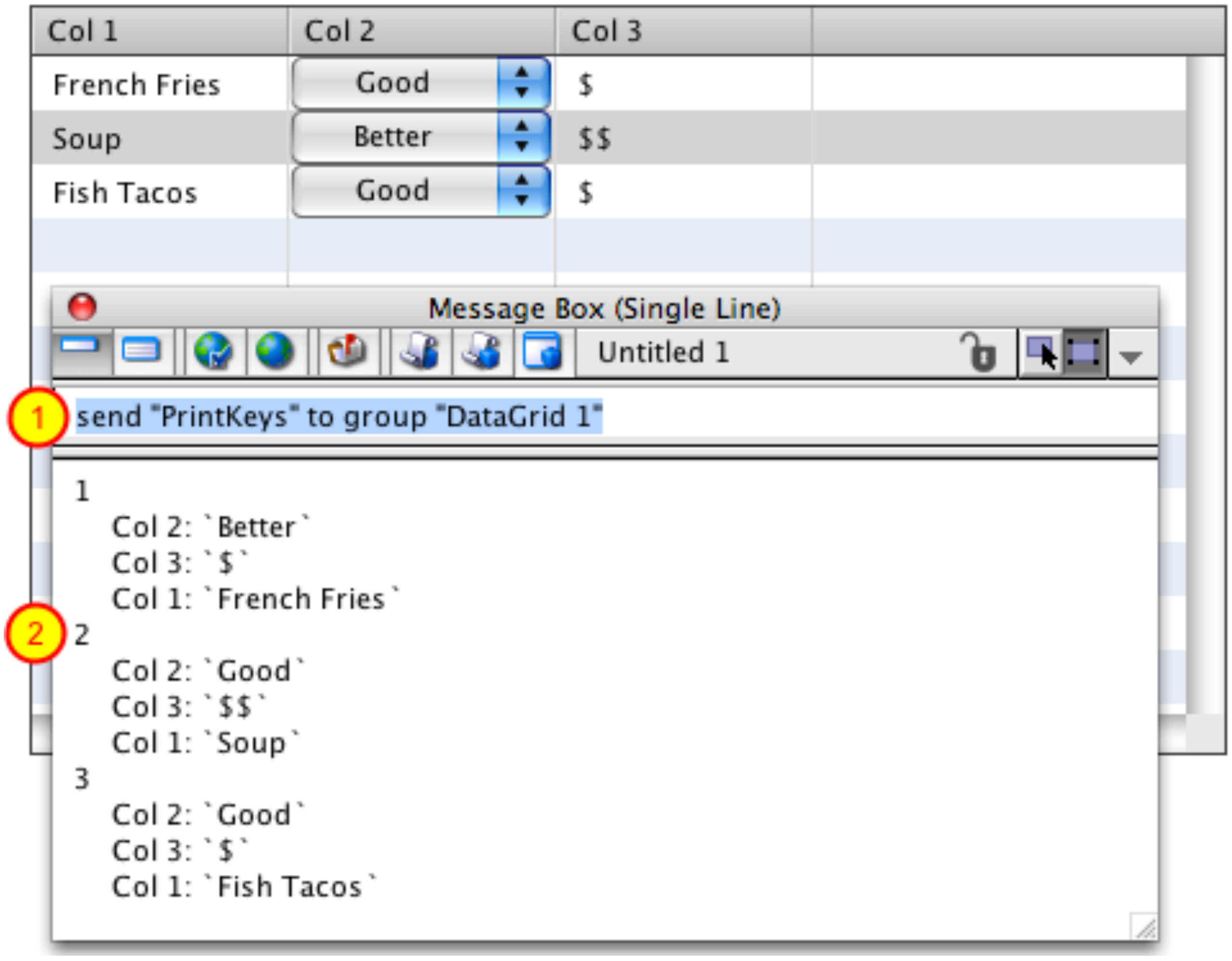
## The Result



Now both data grids will display data using the same template.

## How Can I See What The Data Grid's Internal Array Currently Looks Like?

### Using PrintKeys



```
send "PrintKeys" to group "DataGrid 1"
```

```
1
    Col 2: `Better`
    Col 3: `$`
    Col 1: `French Fries`
2
    Col 2: `Good`
    Col 3: `$$`
    Col 1: `Soup`
3
    Col 2: `Good`
    Col 3: `$`
    Col 1: `Fish Tacos`
```

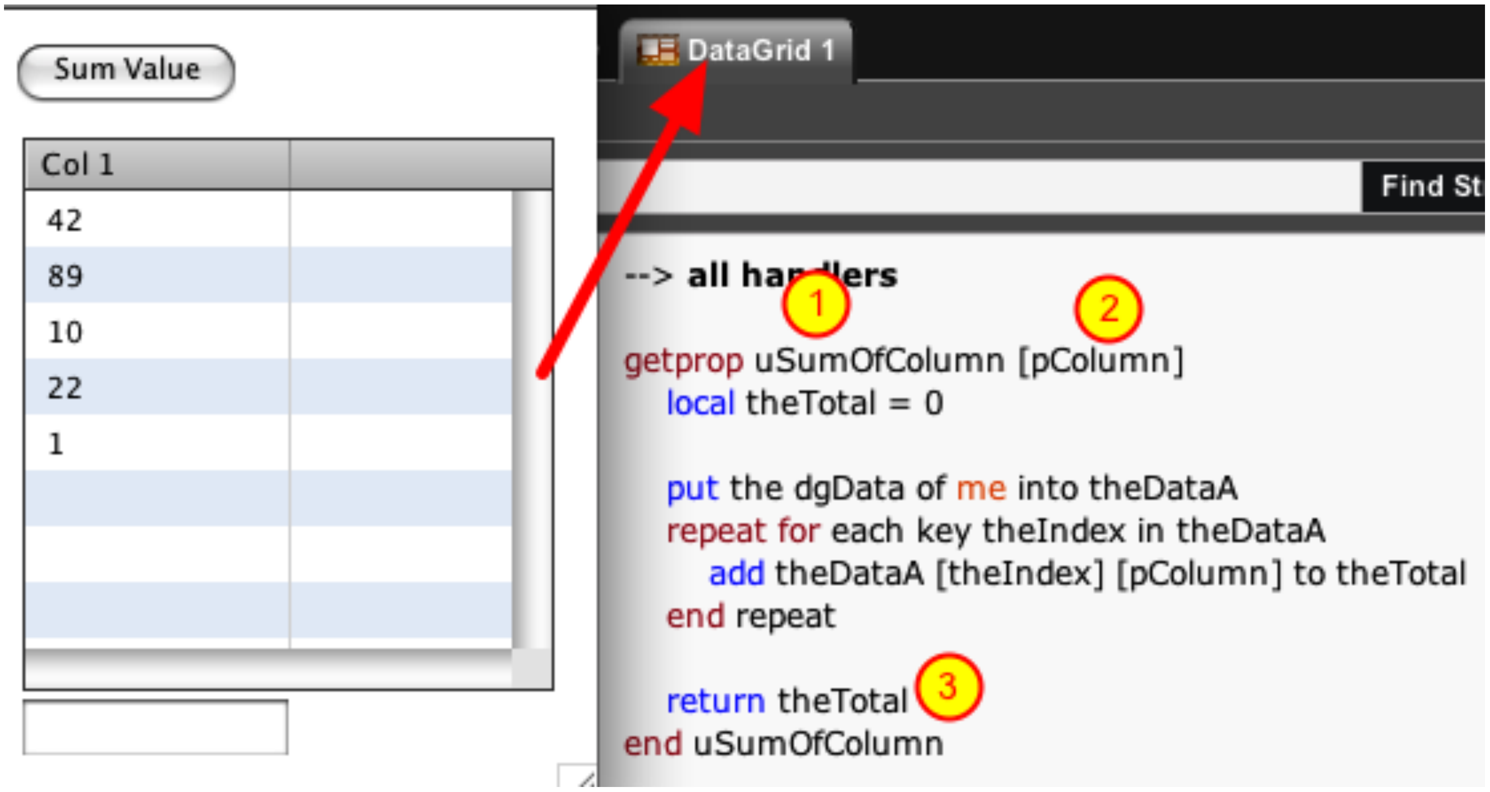
A data grid has a helper command that will print off the first line of each key in the internal data array. If you need to quickly see what the internal array looks like just send "PrintKeys" to the data grid (1). This will print the array in the message box (2).

Note: You should not rely on PrintKeys for anything other than taking a quick peek at the array. It only prints the first line of each key so it is not a 100% accurate view of the data.

## How Do I Get Aggregate Values for Columns?

This lesson will show you how to add a custom property to a data grid that returns the aggregate value of a column. Usually this would be used with data grid tables but works equally as well with data grid forms.

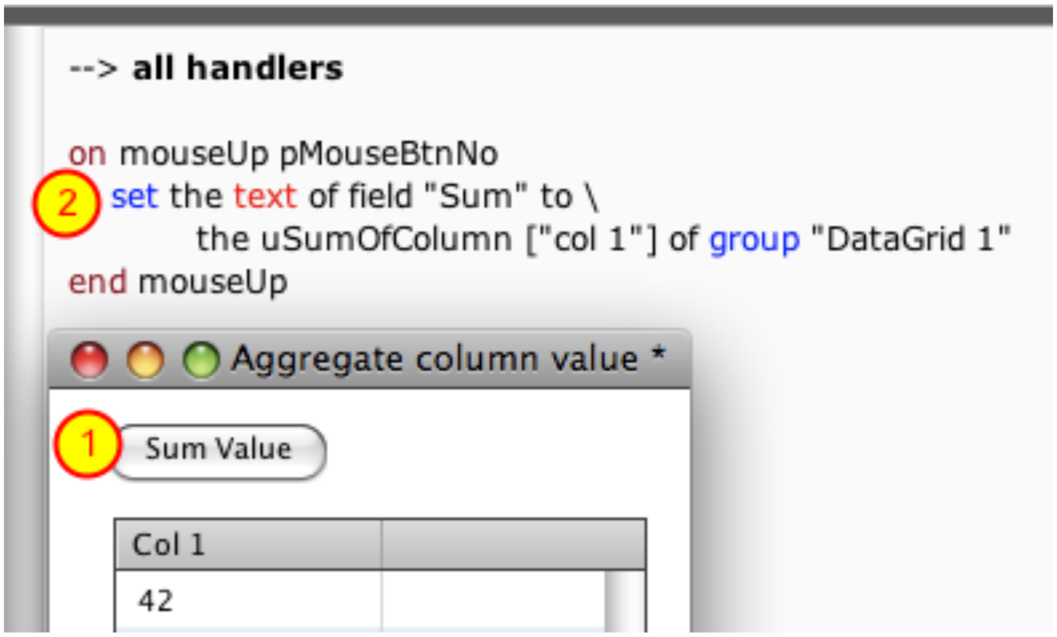
### Add a getProp To Your Data Grid Script



```
--> all handlers

getprop uSumOfColumn [pColumn]
   local theTotal = 0

   put the dgData of me into theDataA
   repeat for each key theIndex in theDataA
      add theDataA [theIndex] [pColumn] to theTotal
   end repeat

   return theTotal
end uSumOfColumn
```

The simplest way to calculate aggregate values is to add a getProp handler to your data grid script. In this example I've defined a custom property called **uSumOfColumn** (1). You can pass in a column name (2) and the sum of all rows of that column will be returned (3).

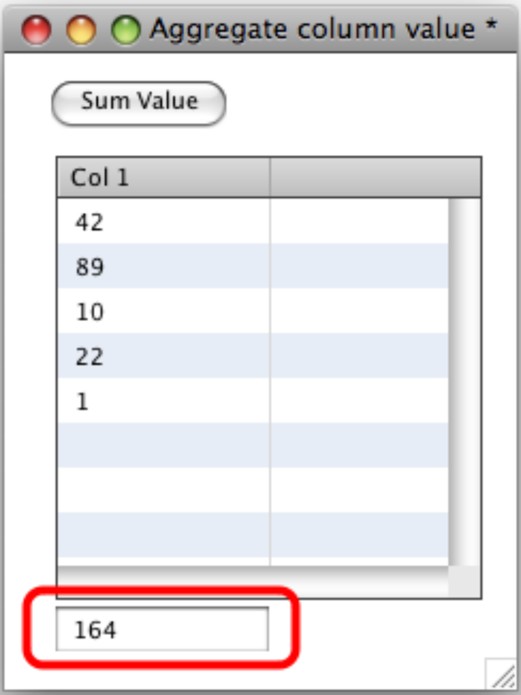
## Using the Custom Property



```
--> all handlers

on mouseUp pMouseBtnNo
   set the text of field "Sum" to \
         the uSumOfColumn ["col 1"] of group "DataGrid 1"
end mouseUp
```

Here is an example of using the custom property. When clicking on the button (1) the text of another field is set to the **uSumOfColumn** custom property (2).

## The Result



| Col 1 | |
|---|---|
| 42 | |
| 89 | |
| 10 | |
| 22 | |
| 1 | |

164

Here the result has been put into the "Sum" field.

## How Do I Determine If the Data Grid Has Focus?

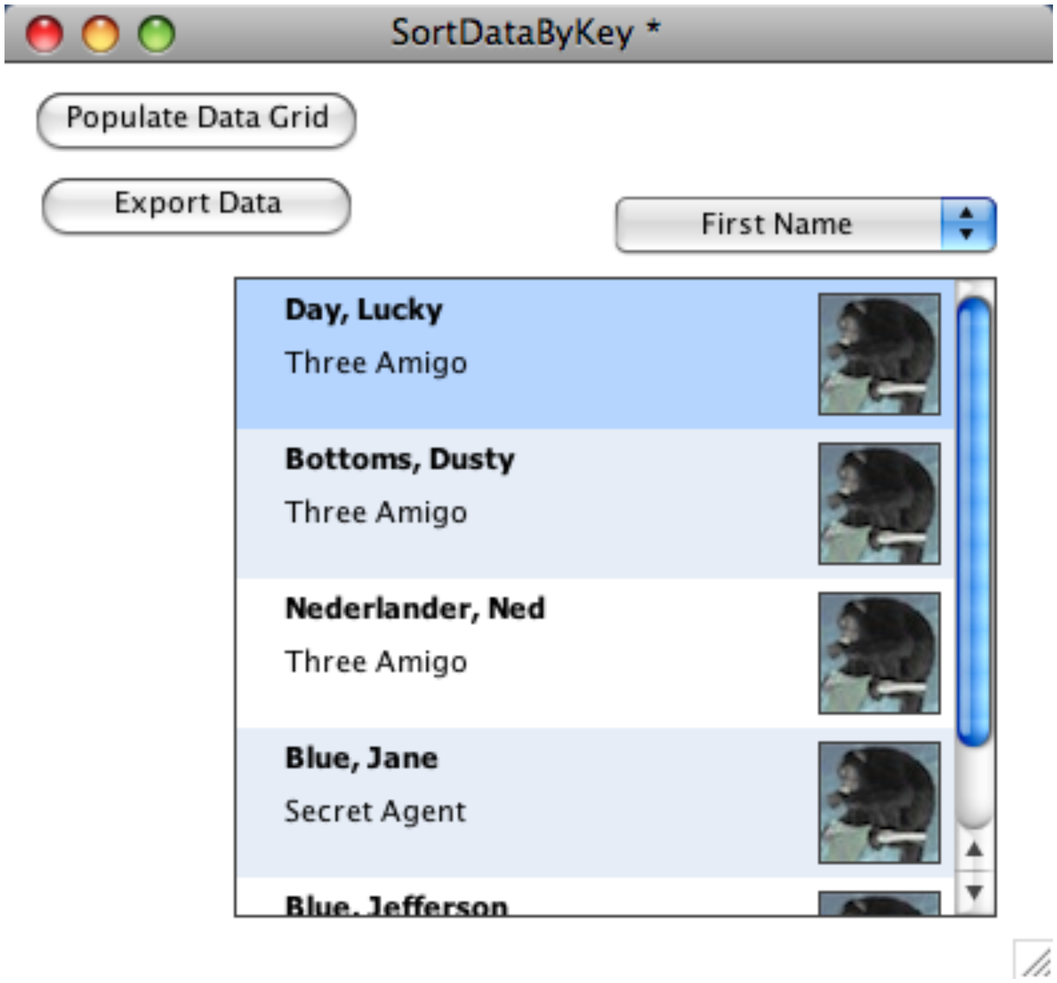
You can determine if the data grid has focus by performing the following check:

```
if the long id of group "DataGrid" is in the long id of the focusedobject then
   ## Data grid has focus
end if
```

## How Do I Export Data From A Data Grid?

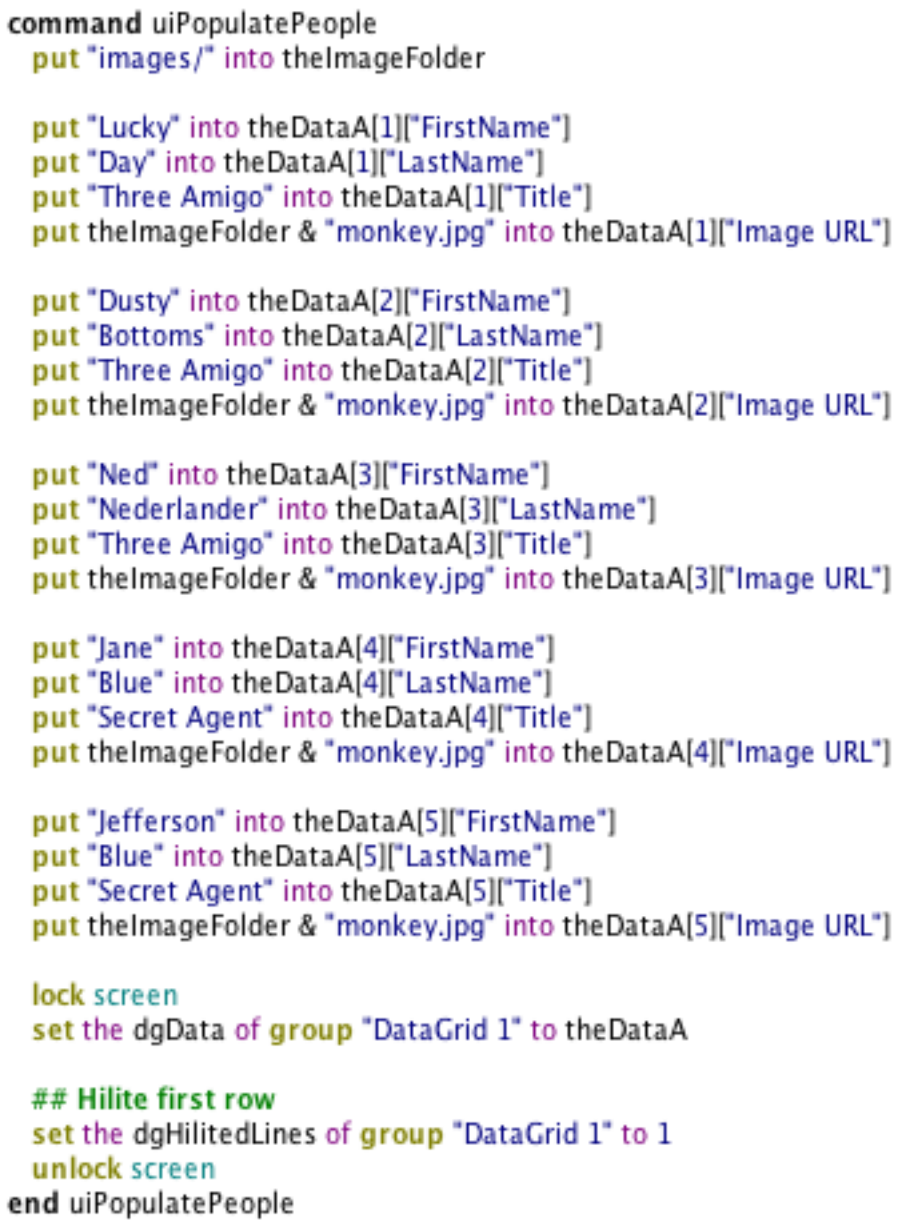
This lesson will show you how to get data out of a data grid.

### The Example Data Grid



Here is what the data grid looks like that I will be exporting data from.

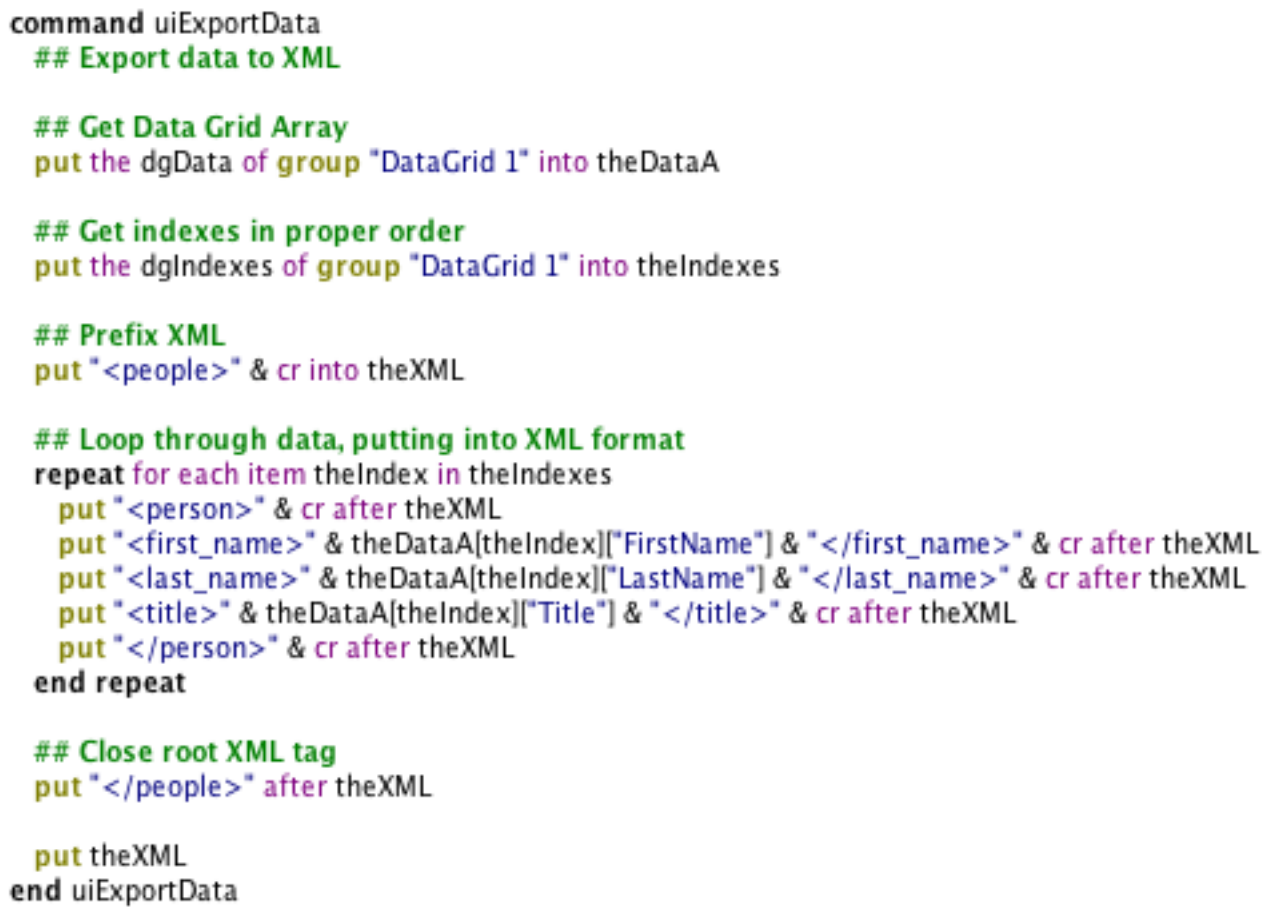
## The Handler For Populating Data Grid



```
command uiPopulatePeople
  put "images/" into theImageFolder

  put "Lucky" into theDataA[1]["FirstName"]
  put "Day" into theDataA[1]["LastName"]
  put "Three Amigo" into theDataA[1]["Title"]
  put theImageFolder & "monkey.jpg" into theDataA[1]["Image URL"]

  put "Dusty" into theDataA[2]["FirstName"]
  put "Bottoms" into theDataA[2]["LastName"]
  put "Three Amigo" into theDataA[2]["Title"]
  put theImageFolder & "monkey.jpg" into theDataA[2]["Image URL"]

  put "Ned" into theDataA[3]["FirstName"]
  put "Nederlander" into theDataA[3]["LastName"]
  put "Three Amigo" into theDataA[3]["Title"]
  put theImageFolder & "monkey.jpg" into theDataA[3]["Image URL"]

  put "Jane" into theDataA[4]["FirstName"]
  put "Blue" into theDataA[4]["LastName"]
  put "Secret Agent" into theDataA[4]["Title"]
  put theImageFolder & "monkey.jpg" into theDataA[4]["Image URL"]

  put "Jefferson" into theDataA[5]["FirstName"]
  put "Blue" into theDataA[5]["LastName"]
  put "Secret Agent" into theDataA[5]["Title"]
  put theImageFolder & "monkey.jpg" into theDataA[5]["Image URL"]

  lock screen
  set the dgData of group "DataGrid 1" to theDataA

  ## Hilite first row
  set the dgHilitedLines of group "DataGrid 1" to 1
  unlock screen
end uiPopulatePeople
```

This is the code that was used to populate the data grid. This shows you the keys that each record has (FirstName, LastName, Title and Image URL).

## Export Handler

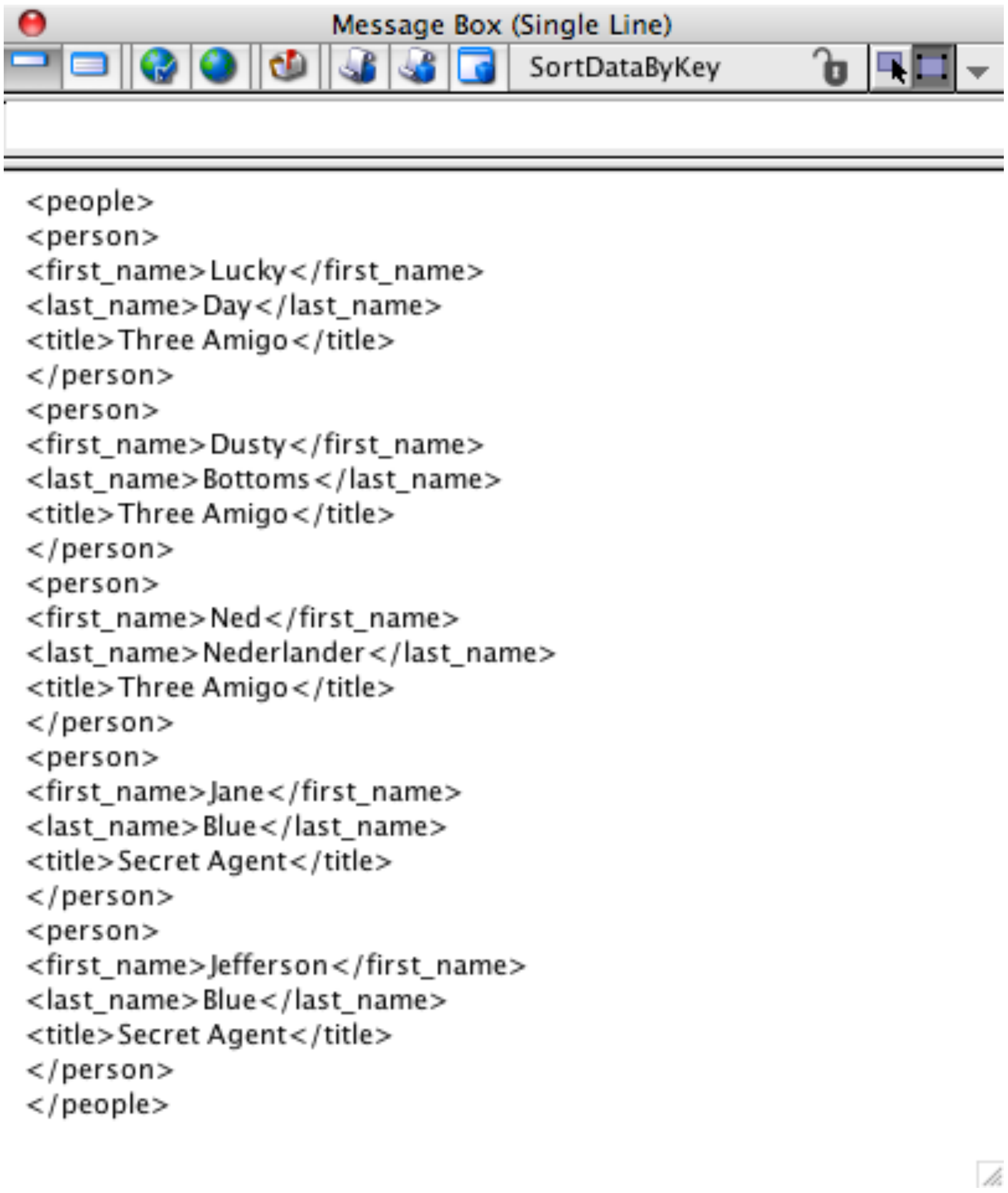


```
command uiExportData
  ## Export data to XML

  ## Get Data Grid Array
  put the dgData of group "DataGrid 1" into theDataA

  ## Get indexes in proper order
  put the dgIndexes of group "DataGrid 1" into theIndexes

  ## Prefix XML
  put "<people>" & cr into theXML

  ## Loop through data, putting into XML format
  repeat for each item theIndex in theIndexes
    put "<person>" & cr after theXML
    put "<first_name>" & theDataA[theIndex]["FirstName"] & "</first_name>" & cr after theXML
    put "<last_name>" & theDataA[theIndex]["LastName"] & "</last_name>" & cr after theXML
    put "<title>" & theDataA[theIndex]["Title"] & "</title>" & cr after theXML
    put "</person>" & cr after theXML
  end repeat

  ## Close root XML tag
  put "</people>" after theXML

  put theXML
end uiExportData
```

This is an example of how to get data out of a data grid. I begin by getting the dgData array and the dgIndexes. The indexes are a comma delimited list of the keys of the dgData in the proper order.

After you have the array and the ordered list of indexes you can loop through each record in the array. In this example I'm just wrapping the data in XML tags.

## Example Output



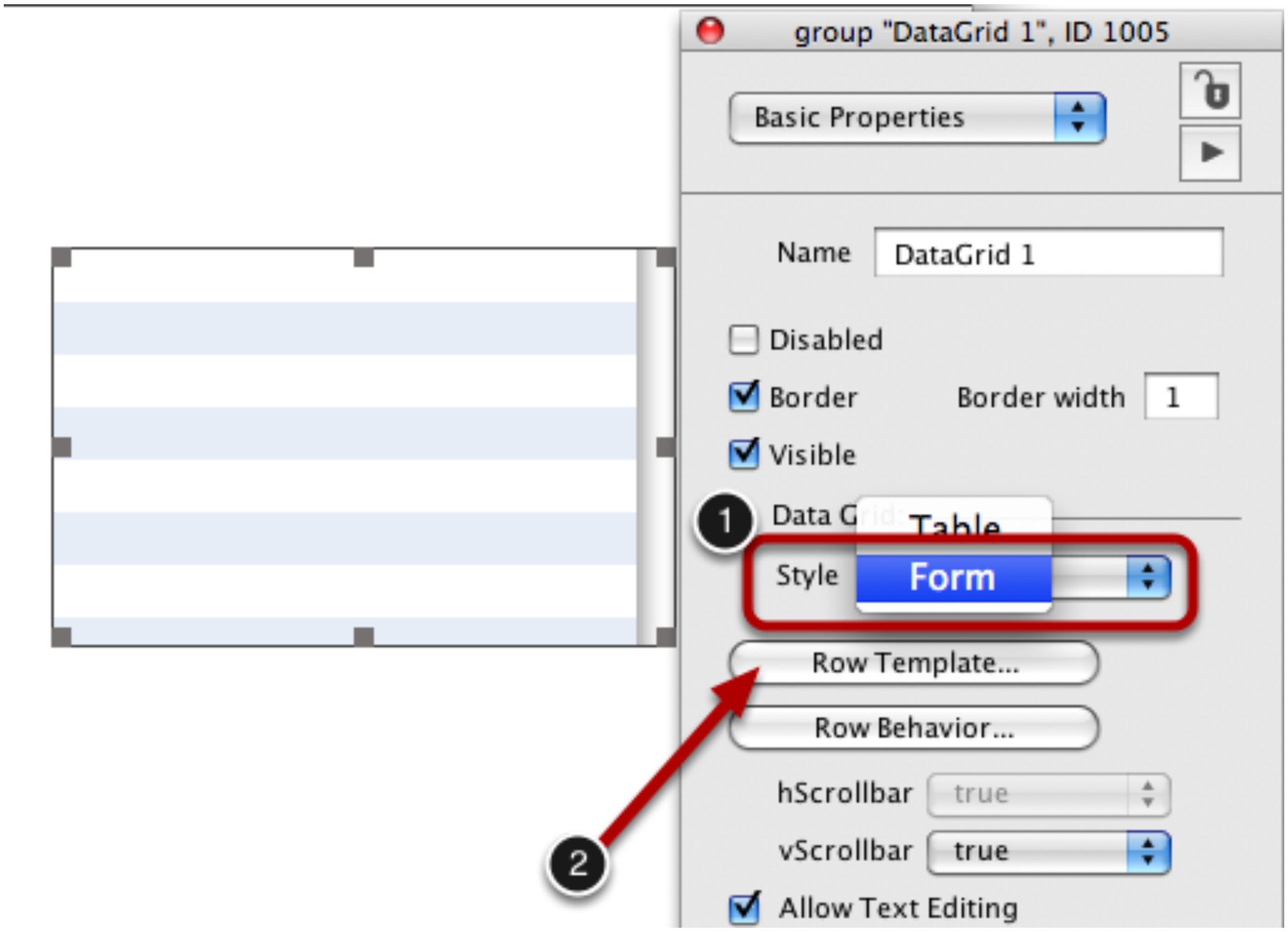
```
<people>
<person>
<first_name>Lucky</first_name>
<last_name>Day</last_name>
<title>Three Amigo</title>
</person>
<person>
<first_name>Dusty</first_name>
<last_name>Bottoms</last_name>
<title>Three Amigo</title>
</person>
<person>
<first_name>Ned</first_name>
<last_name>Nederlander</last_name>
<title>Three Amigo</title>
</person>
<person>
<first_name>Jane</first_name>
<last_name>Blue</last_name>
<title>Secret Agent</title>
</person>
<person>
<first_name>Jefferson</first_name>
<last_name>Blue</last_name>
<title>Secret Agent</title>
</person>
</people>
```

This is what the output for my example looks like in the message box.

## How Do I Work with Checkboxes in a Data Grid?

This lesson will demonstrate how to associate the hilite state of a checkbox in a data grid with a row value in the data grid.

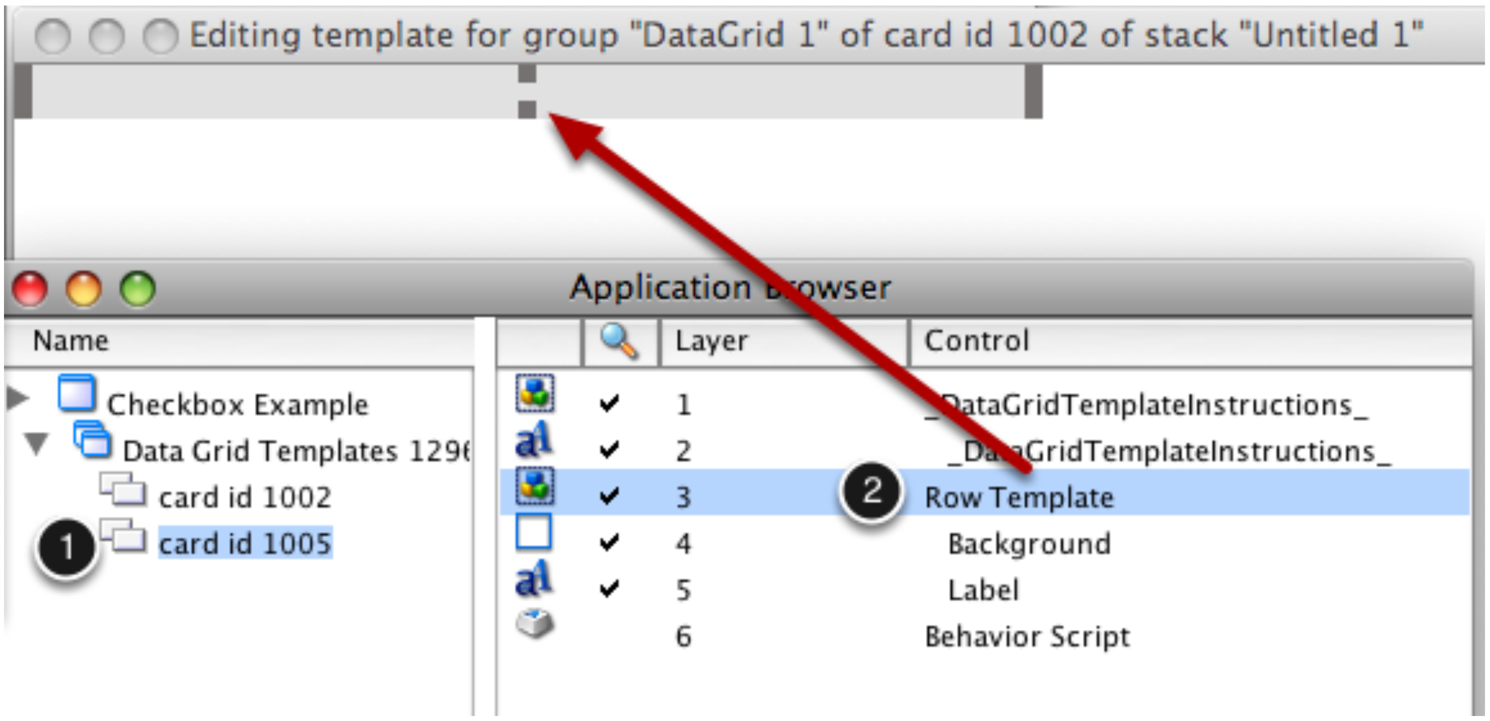
### Adding a Checkbox to a Form



Be begin, drag a data grid onto a stack and set the style to **Form** using the Object Inspector.

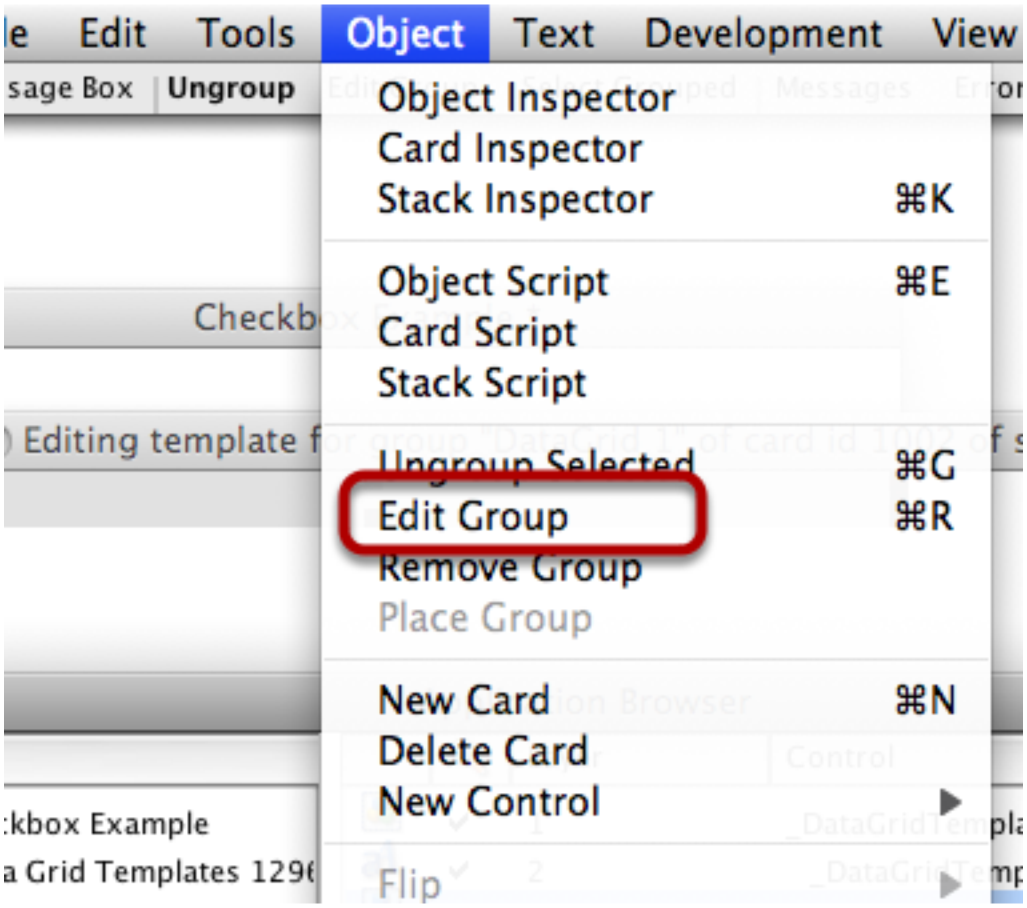
Now we can add the checkbox. Click the **Row Template** button to open the template for the data grid.

## Edit Row Template Group



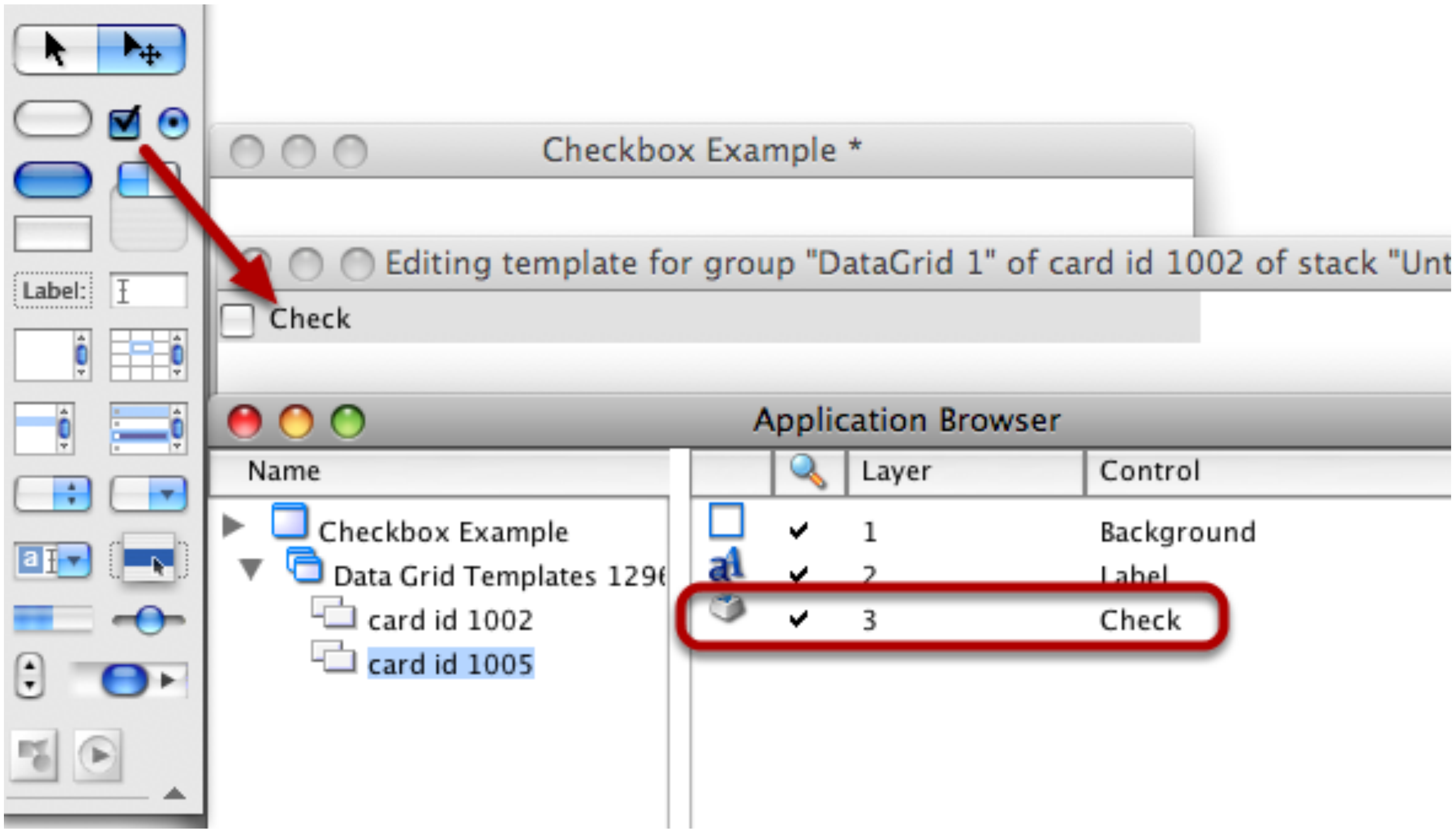
Now that the card with the data grid row template is visible, open the **Application Browser** selecting the **Tools > Application Browser** menu option.

Select the card with your template (it should be the second card under the Data Grid Templates stack) (1). Next, select the **Row Templat**e group on the card (2). Doing so will select the group on the card.



Now select the **Object > Edit Group** menu option. This will put the group in edit mode which will allow you to drag controls from the tools palette into the group.

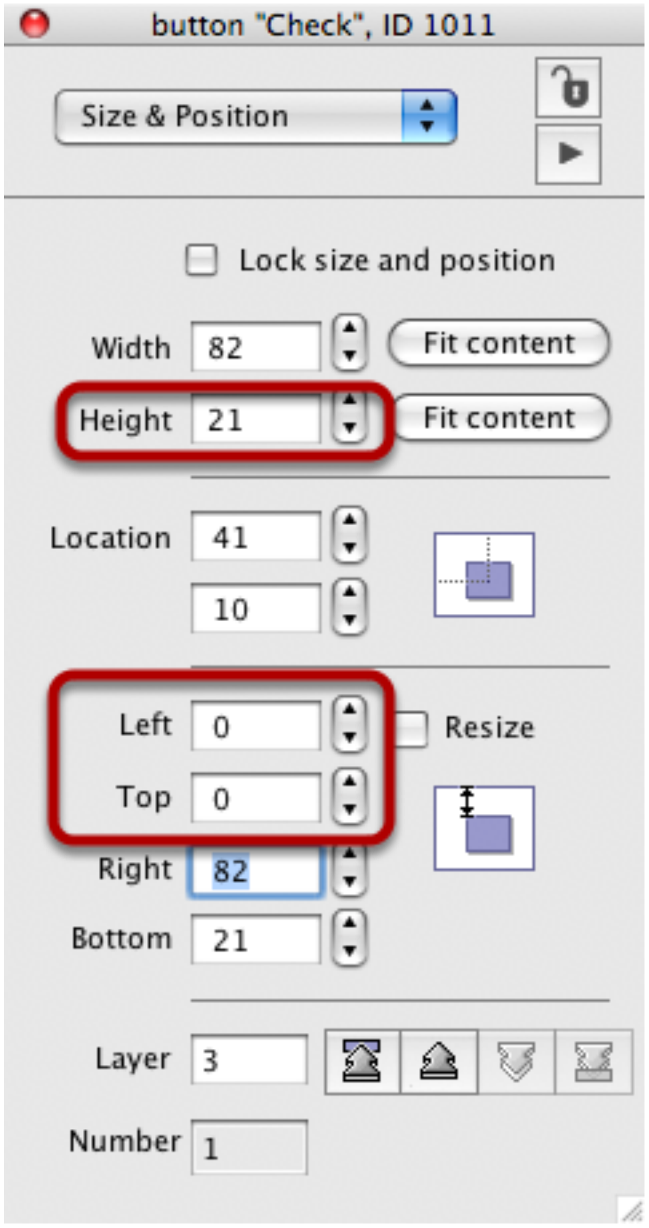
## Add a Checkbox



|  |  | Layer | Control |
|---|---|---|---|
|  | ✓ | 1 | Background |
|  | ✓ | 2 | Label |
|  | ✓ | 3 | Check |

From the tools palette, drag a checkbox onto the card and place it in the upper-left corner. The button will appear in the list of controls in the Application Browser.

## Edit Checkbox Properties


button "Check", ID 1011

Size & Position

Lock size and position

Width 82

Height 21

Location 41, 10

Left 0

Top 0

Right 82

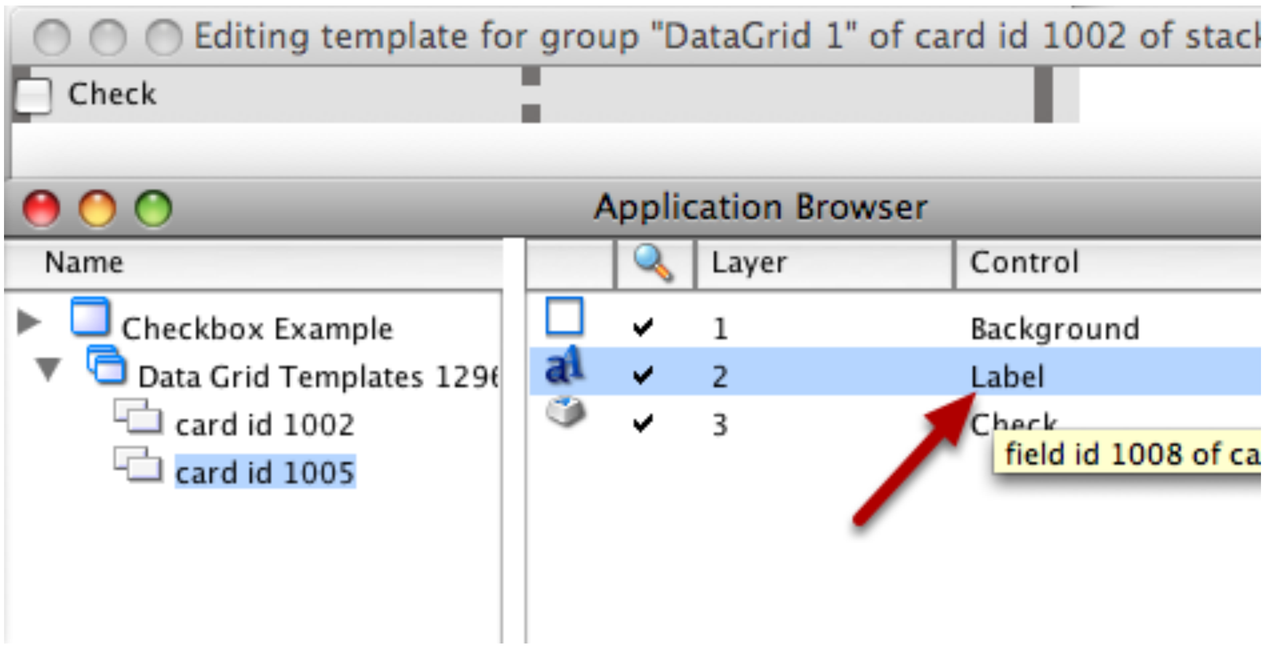
Bottom 21

Layer 3

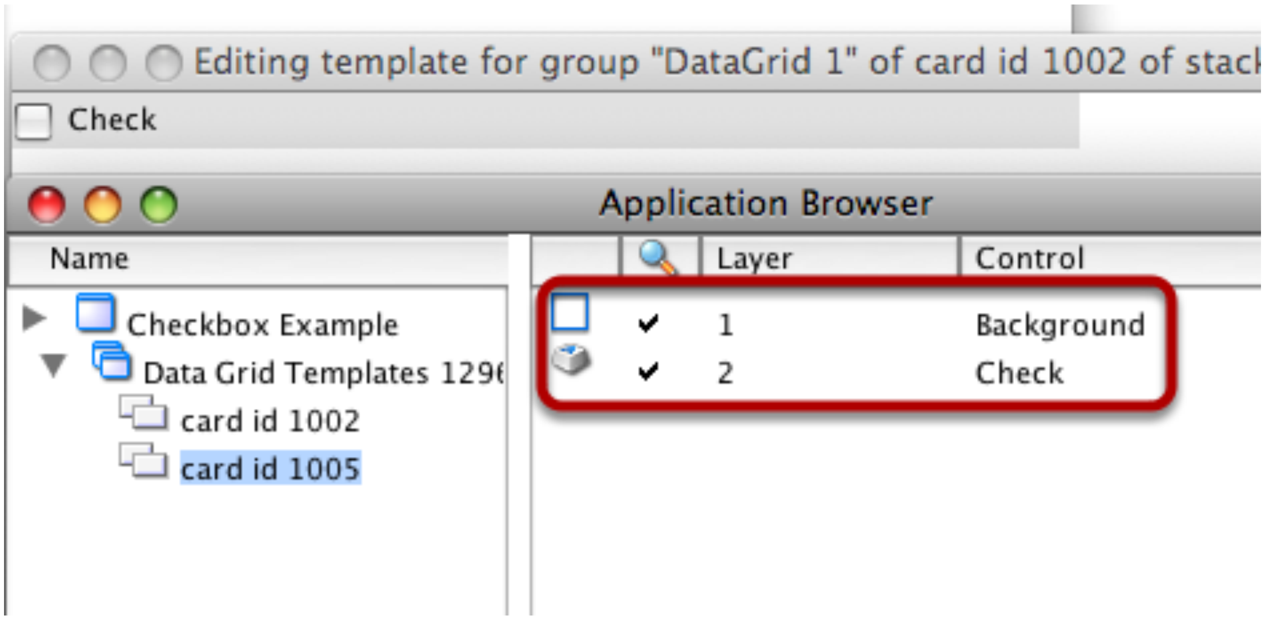
Number 1

Open the Object Inspector for the button and change the **height** to 21. Set the **left** and **top** properties to 0.

## Delete the Label Field

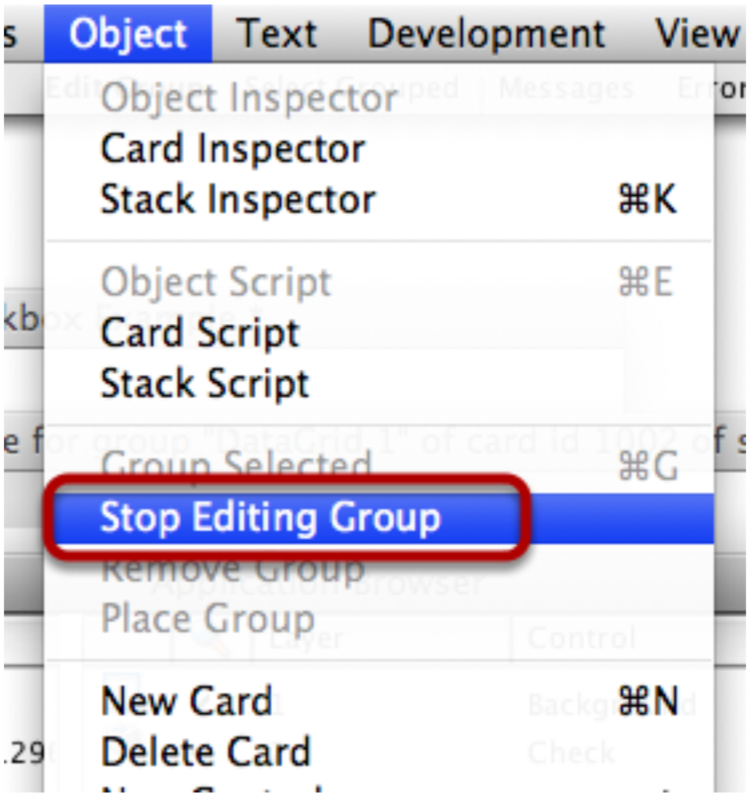


Now that you have added the checkbox button you no longer need the **Label** field in the row template. Select the **Label** field and delete it.



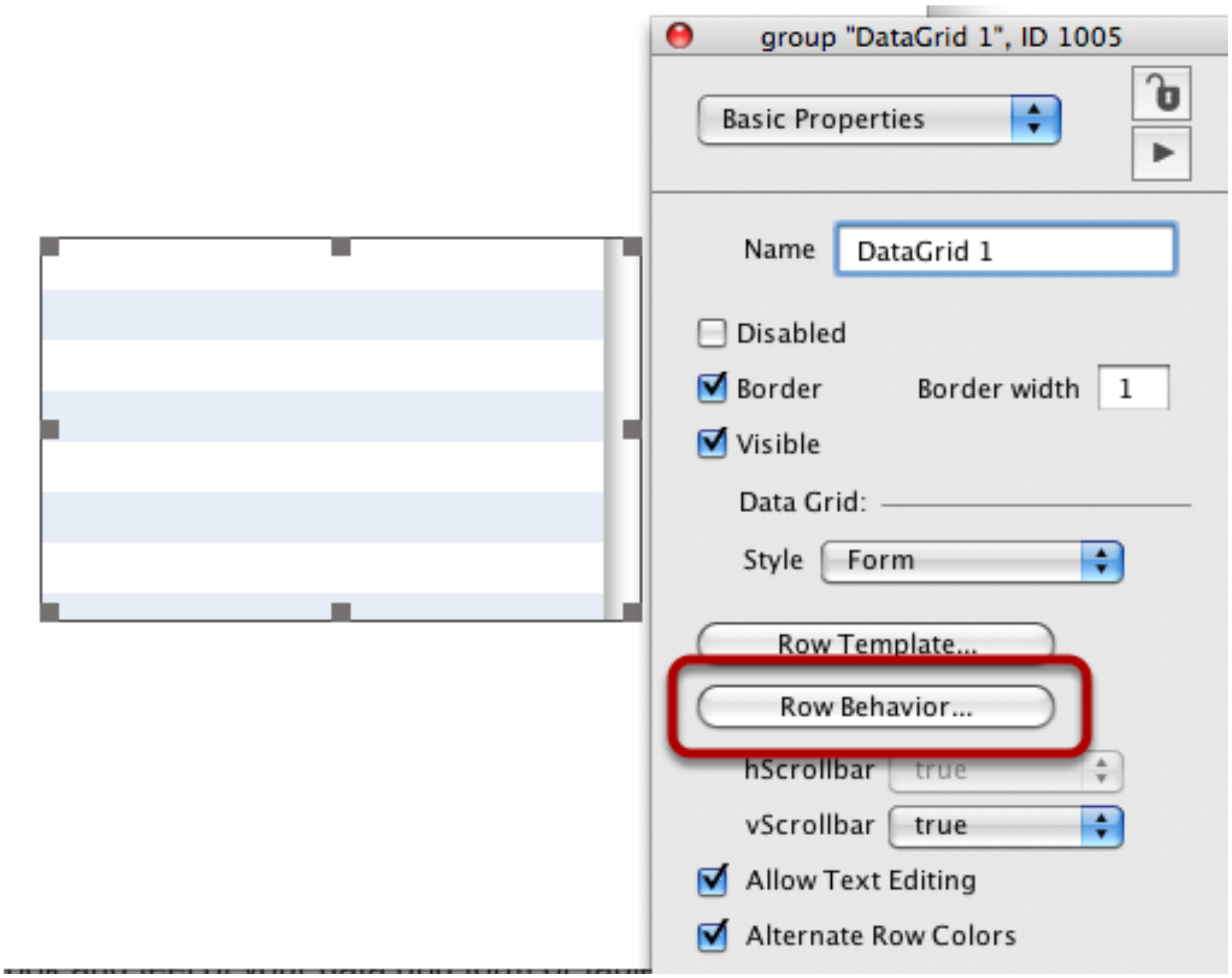
After deleting the field you should have the Background graphic and the Check checkbox in the group. Remember, you are still editing the group so the group is not listed in the card controls at the moment.

## Stop Editing Group



You are now done editing the group controls so select the **Object > Stop Editing Group** menu option.

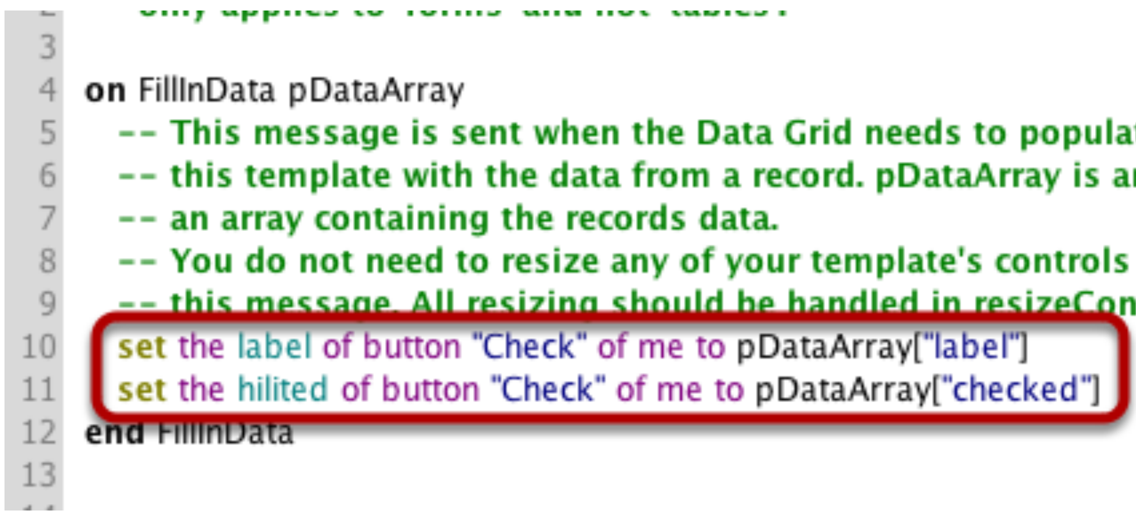
## Edit Row Behavior



Next you need to edit the **Row Behavior** in order to take into account the new checkbox control.

Click the **Row Behavior** button.

## Edit the FillInData Handler



```
on FillInData pDataArray
  -- This message is sent when the Data Grid needs to populate
  -- this template with the data from a record. pDataArray is an
  -- an array containing the records data.
  -- You do not need to resize any of your template's controls in
  -- this message. All resizing should be handled in resizeControl.
  set the label of button "Check" of me to pDataArray["label"]
  set the hilited of button "Check" of me to pDataArray["checked"]
end FillInData
```

In the FillInData Handler you need to remove references to the **Label** field and add code for the **Check** button. The code will set the **label** of the checkbox to the "label" property of the data grid row. It will set the **hilited** property to the "checked" property of the row.

**----------**
**Copy & Paste**
**----------**

```
on FillInData pDataArray
   -- This message is sent when the Data Grid needs to populate
   -- this template with the data from a record. pDataArray is an
   -- an array containing the records data.
   -- You do not need to resize any of your template's controls in
   -- this message. All resizing should be handled in resizeControl.
   set the label of button "Check" of me to pDataArray["label"]
   set the hilited of button "Check" of me to pDataArray["checked"]
end FillInData
```

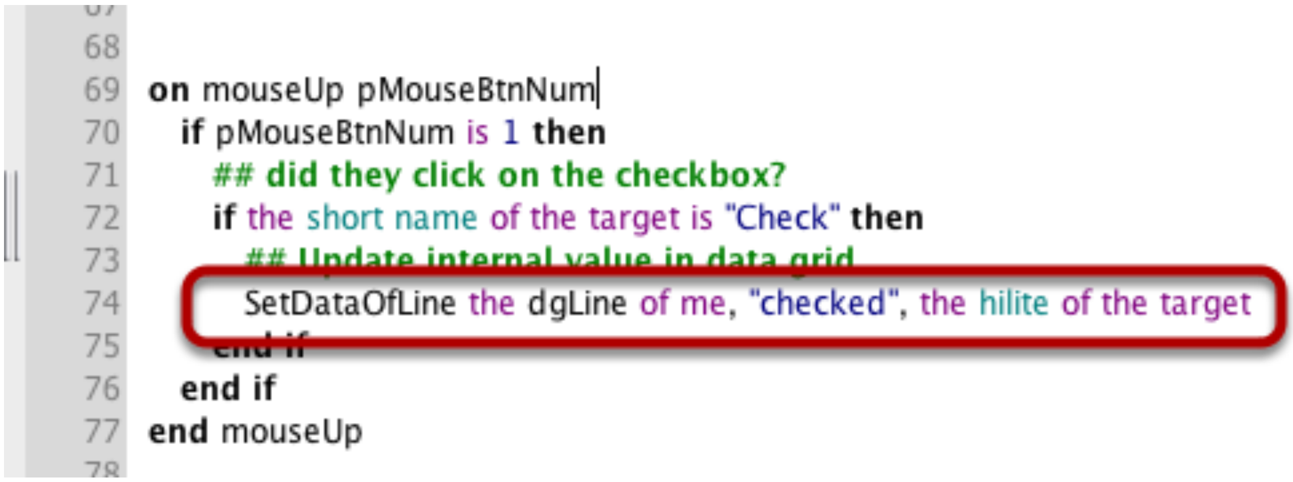
## Update LayoutControl and ResetData

The LayoutControl and ResetData handlers also have references to the **Label** field which no longer exists. Update both of those handlers with references to the **Check** button.

**----------**
**Copy & Paste**
**----------**

```
on LayoutControl pControlRect
   local theFieldRect
   -- This message is sent when you should layout your template's controls.
   -- This is where you resize the 'Background' graphic, resize fields and
   -- position objects.
   -- For fixed height data grid forms you can use items 1 through 4 of pControlRect as
   -- boundaries for laying out your controls.
   -- For variable height data grid forms you can use items 1 through 3 of pControlRect as
   -- boundaries, expanding the height of your control as needed.

   -- Example:
   put the rect of button "Check" of me into theFieldRect
   put item 3 of pControlRect - 5 into item 3 of theFieldRect
   set the rect of button "Check" of me to theFieldRect

   set the rect of graphic "Background" of me to pControlRect
end LayoutControl


on ResetData
   -- Sent when data is being emptied because the control is no longer being used to display data
   set the text of button "Check" of me to empty
   set the hilite of button "Check" of me to false
end ResetData
```

## Update Data Grid Row Value When User Changes Checkbox State



```
on mouseUp pMouseBtnNum
  if pMouseBtnNum is 1 then
    ## did they click on the checkbox?
    if the short name of the target is "Check" then
      ## Update internal value in data grid
      SetDataOfLine the dgLine of me, "checked", the hilite of the target
    end if
  end if
end mouseUp
```

When the user clicks on the checkbox the hilited state will change. You need to update the row data in the data grid when this happens so that the dgData array properly represents what is being seen in the interface. You can do this in the mouseUp behavior of the behavior script.

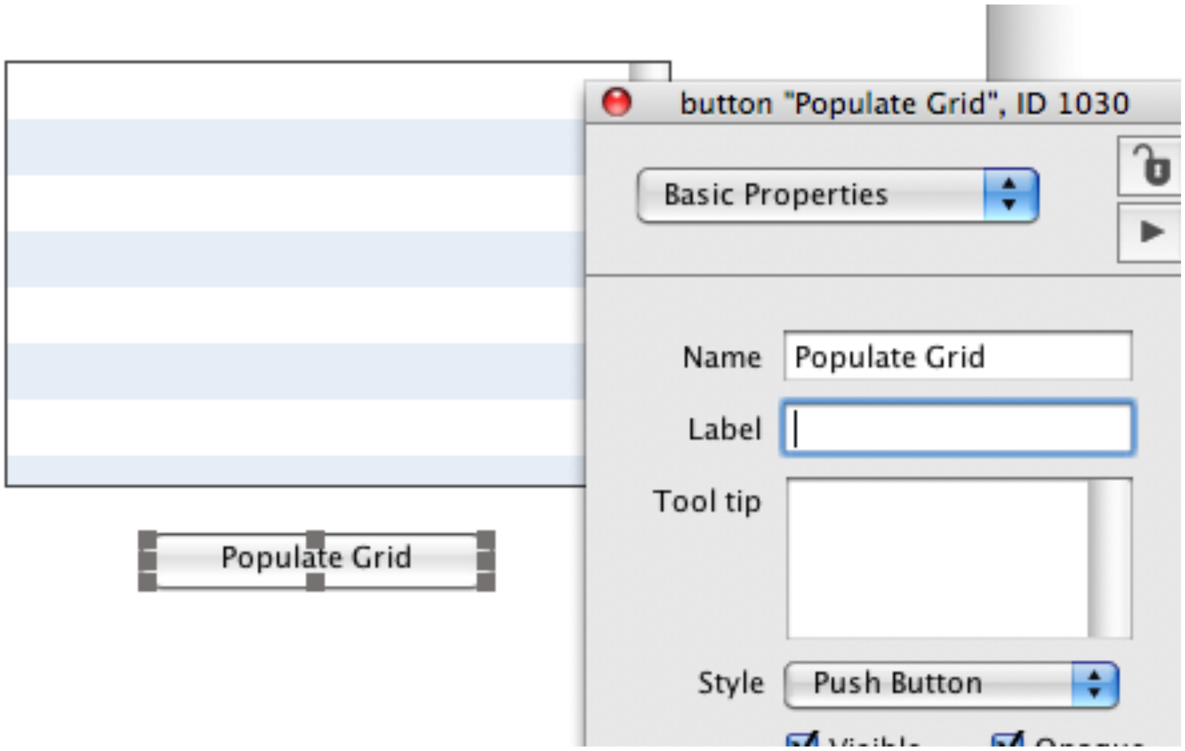
During mouseUp the hilited state of the checkbox has been changed and so it is safe to save the value. You just need to call SetDataOfLine and pass in **the hilite of the target** (the target being the checkbox button).

Add the following to the behavior script. Make sure and compile the script afterwards.

**----------**
**Copy & Paste**
**----------**

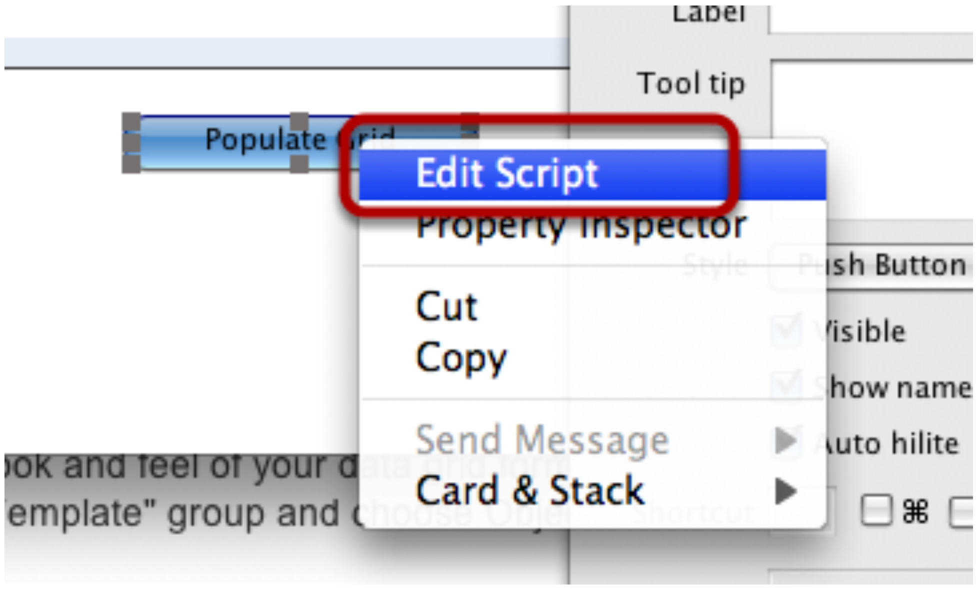
```
on mouseUp pMouseBtnNum
   if pMouseBtnNum is 1 then
      ## did they click on the checkbox?
      if the short name of the target is "Check" then
         ## Update internal value in data grid
         SetDataOfLine the dgLine of me, "checked", the hilite of the target
      end if
   end if
end mouseUp
```

## Populate the Data Grid

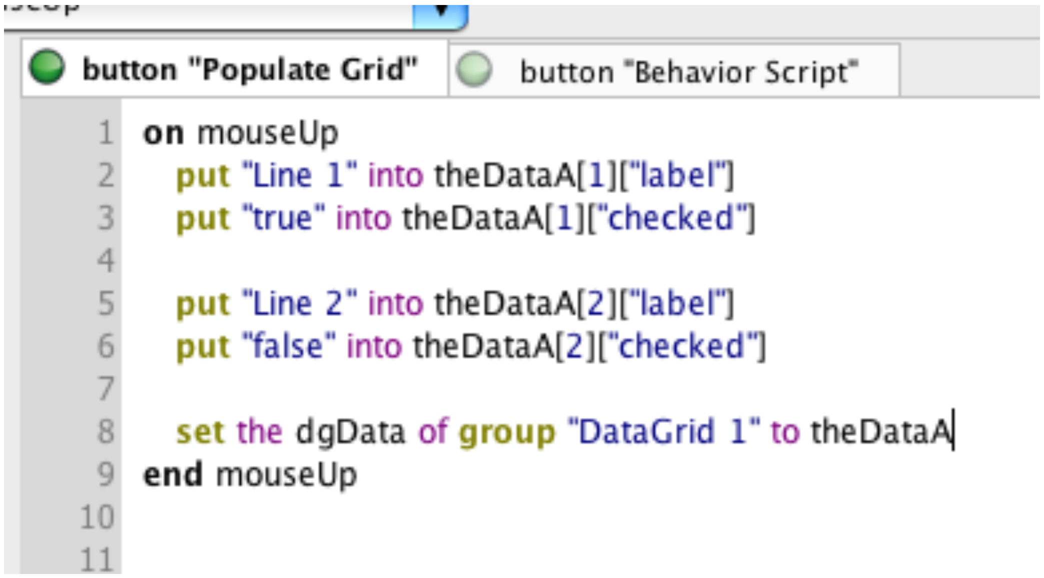


Name: Populate Grid
Label:
Tool tip:
Style: Push Button

You have now finished configuring the Data Grid so it is time to test. Add a button to the stack and name it **Populate grid**.



With the Edit tool active, right-click on the button and select **Edit Script**.



```
on mouseUp
   put "Line 1" into theDataA[1]["label"]
   put "true" into theDataA[1]["checked"]

   put "Line 2" into theDataA[2]["label"]
   put "false" into theDataA[2]["checked"]

   set the dgData of group "DataGrid 1" to theDataA
end mouseUp
```

Add a simple script that will populate the data grid with two rows. Each has the "label" and "checked" properties that you referenced in the behavior script earlier.

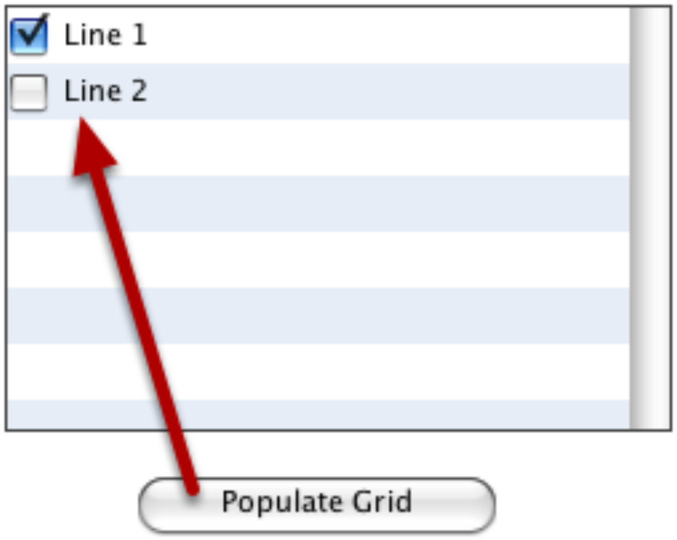
Paste this script to the button script.

**----------**
**Copy & Paste**
**----------**

```
on mouseUp
   put "Line 1" into theDataA[1]["label"]
   put "true" into theDataA[1]["checked"]

   put "Line 2" into theDataA[2]["label"]
```

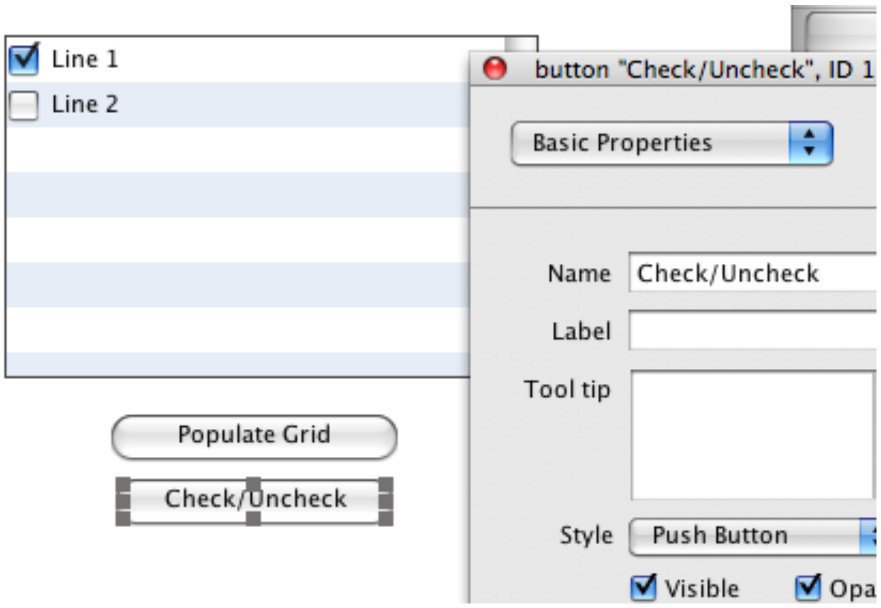
```
    put "false" into theDataA[2]["checked"]

    set the dgData of group "DataGrid 1" to theDataA
end mouseUp
```

## Test



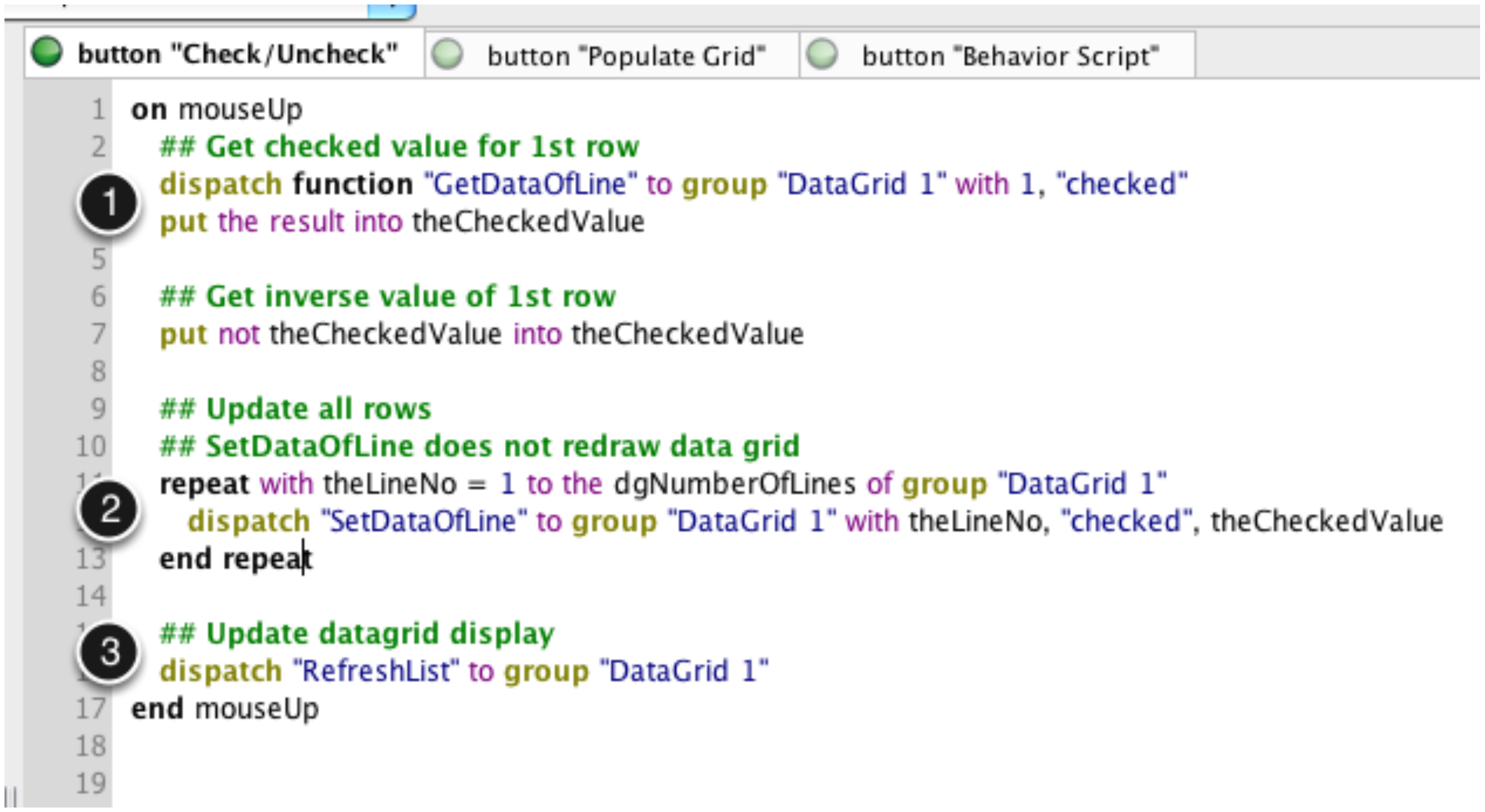
After compiling the script, click on the **Populate Grid** button. Your card should now look similar to this.

## Checking/Unchecking All Checkboxes in a Data Grid



When working with lists that have checkboxes it is nice to include the option to check or uncheck all of the items in the list. Let's look at how to do that.

Add a button to the card and name it **Check/Uncheck**. Edit the script of the button.



```
on mouseUp
  ## Get checked value for 1st row
  dispatch function "GetDataOfLine" to group "DataGrid 1" with 1, "checked"
  put the result into theCheckedValue

  ## Get inverse value of 1st row
  put not theCheckedValue into theCheckedValue

  ## Update all rows
  ## SetDataOfLine does not redraw data grid
  repeat with theLineNo = 1 to the dgNumberOfLines of group "DataGrid 1"
    dispatch "SetDataOfLine" to group "DataGrid 1" with theLineNo, "checked", theCheckedValue
  end repeat

  ## Update datagrid display
  dispatch "RefreshList" to group "DataGrid 1"
end mouseUp
```

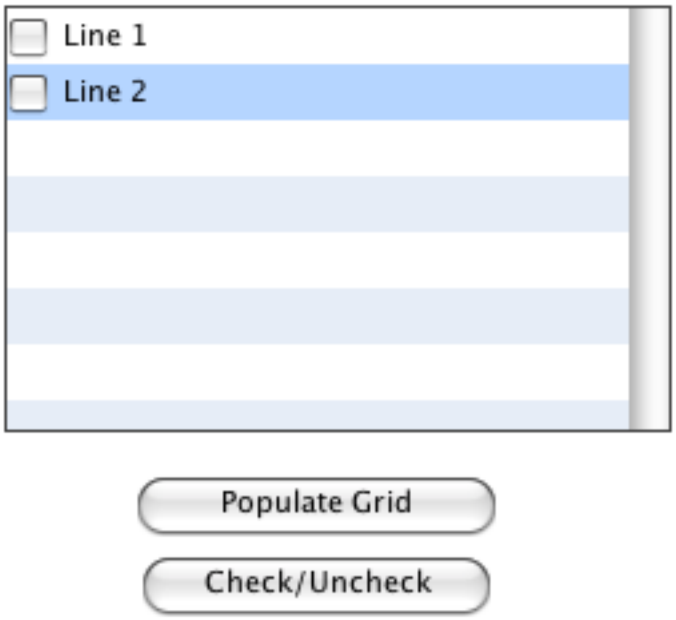
The code for checking/unchecking all items in the list isn't too complicated. Here is how the logic is broken up:

<ol><li>We will use the "checked" value of row 1 to decide what to set all of the other rows to. Get the current value of row 1 and toggle it.
</li><li>Loop through all lines of the data grid, setting the value of the "checked" property of each row to the new value. Use **SetDataOfLine** as this command will not redraw the data grid each time it is called.
</li><li>Refresh the data grid so that the new row values are used to display each row.
</li></ol>
**----------**
**Copy & Paste**
**----------**

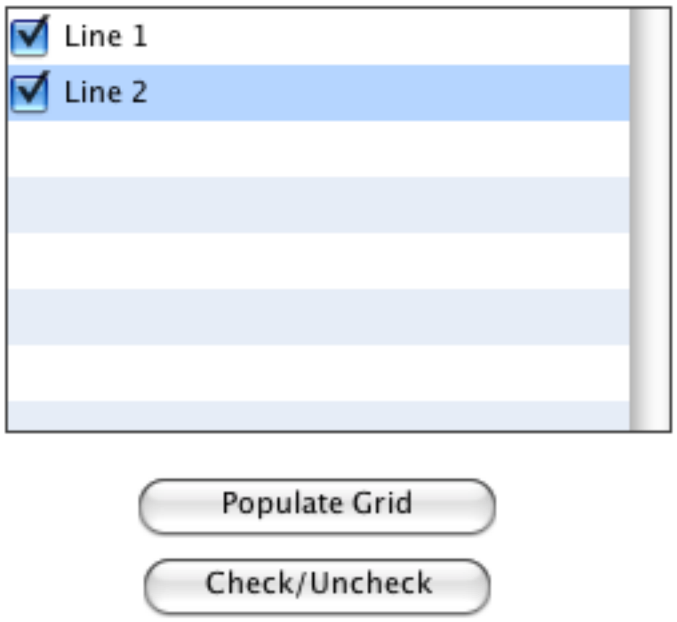
```
on mouseUp
   ## Get checked value for 1st row
   dispatch function "GetDataOfLine" to group "DataGrid 1" with 1, "checked"
   put the result into theCheckedValue

   ## Get inverse value of 1st row
   put not theCheckedValue into theCheckedValue

   ## Update all rows
```

```
    ## SetDataOfLine does not redraw data grid
    repeat with theLineNo = 1 to the dgNumberOfLines of group "DataGrid 1"
       dispatch "SetDataOfLine" to group "DataGrid 1" with theLineNo, "checked", theCheckedValue
    end repeat

    ## Update datagrid display
    dispatch "RefreshList" to group "DataGrid 1"
end mouseUp
```

## Test



Click the **Check/Uncheck** button to test. You should see the checkboxes toggle between the checked and unchecked state.

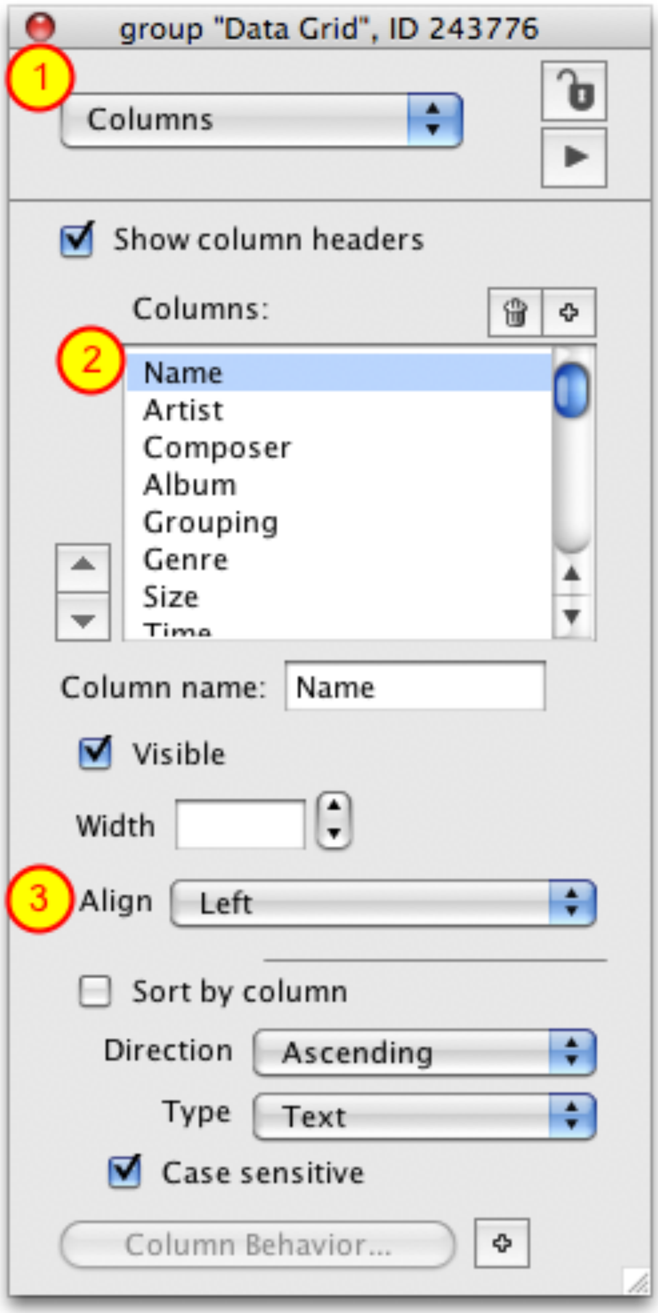


If you get the dgData or dgText property of the data grid then you will see the proper values for the "checked" state of each row.

# Working With Data Grid Tables

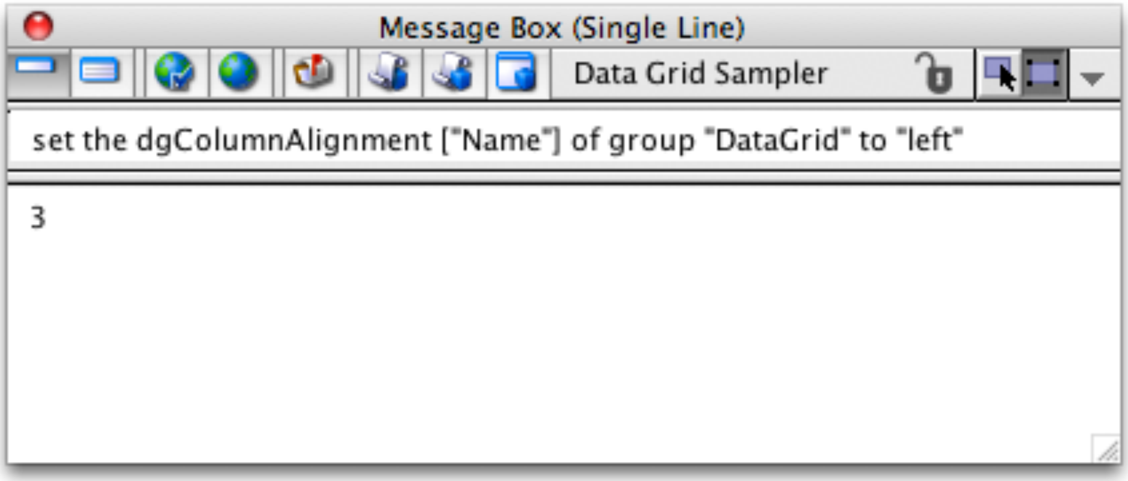
## How Do I Change Column Alignment?

### Using the Property Inspector



Select the data grid and open the property inspector. Navigate to the **Columns** pane (1), select the column you want to modify (2), and change the alignment (3).

## Using Script



```
set the dgColumnAlignment ["Name"] of group "DataGrid" to "left"
```

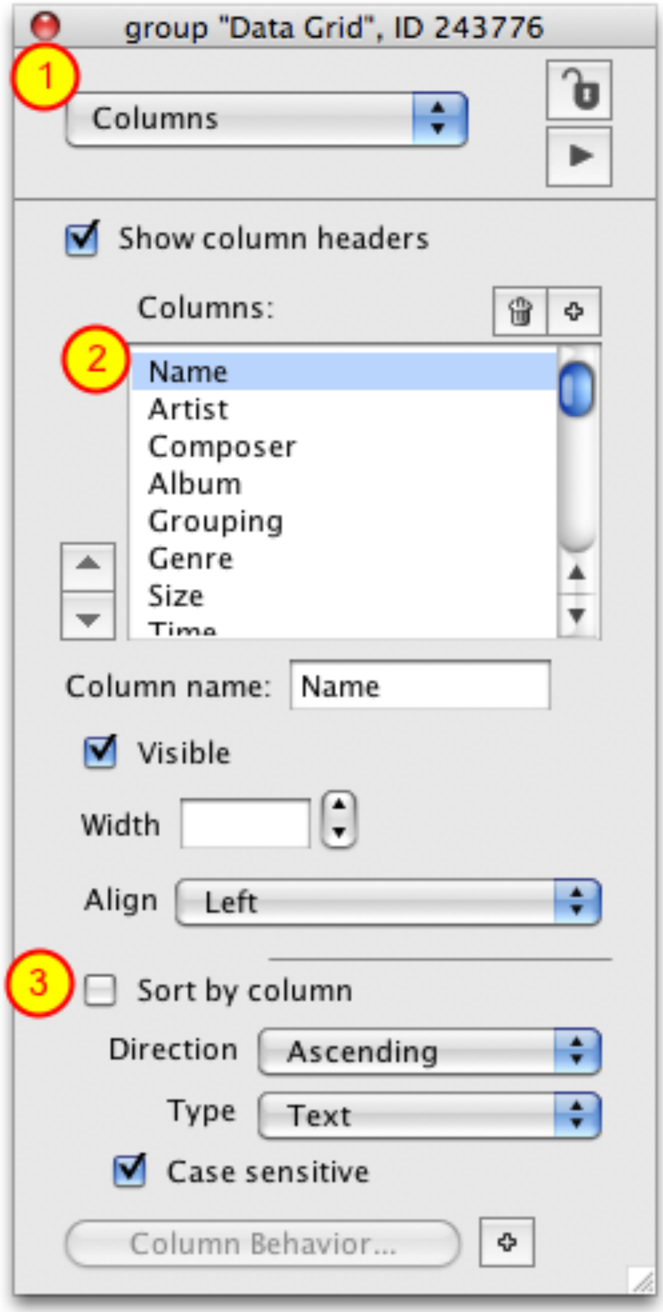
```
3
```

You can change the alignment by setting the dgColumnAlignment for a column.

```
set the dgColumnAlignment["Name"] of group "DataGrid" to "left"
```

## How Do I Sort By A Column?

### Using the Property Inspector



Column name: Name

✓ Visible

Align: Left

☐ Sort by column

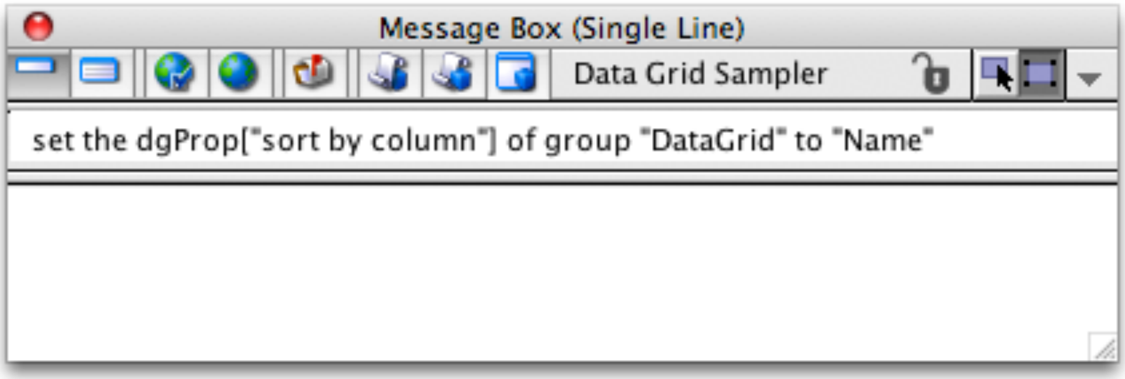
Direction: Ascending

Type: Text

✓ Case sensitive

Select the data grid and open the property inspector. Navigate to the **Columns** pane (1), select the column you want to modify (2), and check the **Sort by column** checkbox (3).

## Using Script



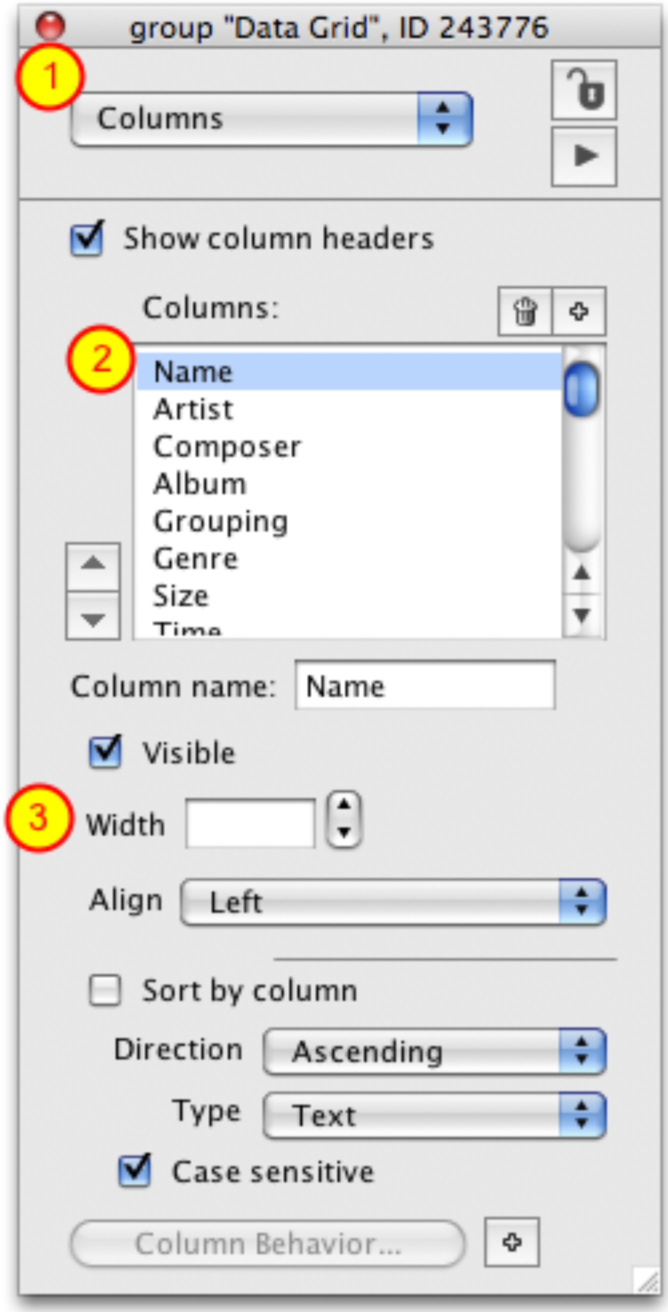
```
set the dgProp["sort by column"] of group "DataGrid" to "Name"
```

You can sort by column setting the dgProp["sort by column"] property of the data grid to the name of the column you want to sort by.

```
set the dgProps["sort by column"] of group "DataGrid" to "Name"
```

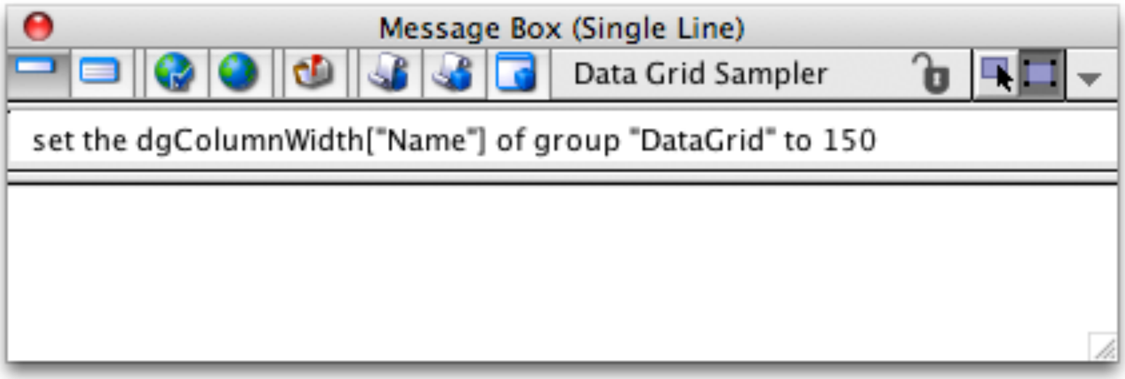
## How Do I Resize Columns?

### Using the Property Inspector



Select the data grid and open the property inspector. Navigate to the **Columns** pane (1), select the column you want to modify (2), and set the width of the column using the Width text entry field (3).

## Using Script



```
set the dgColumnWidth["Name"] of group "DataGrid" to 150
```

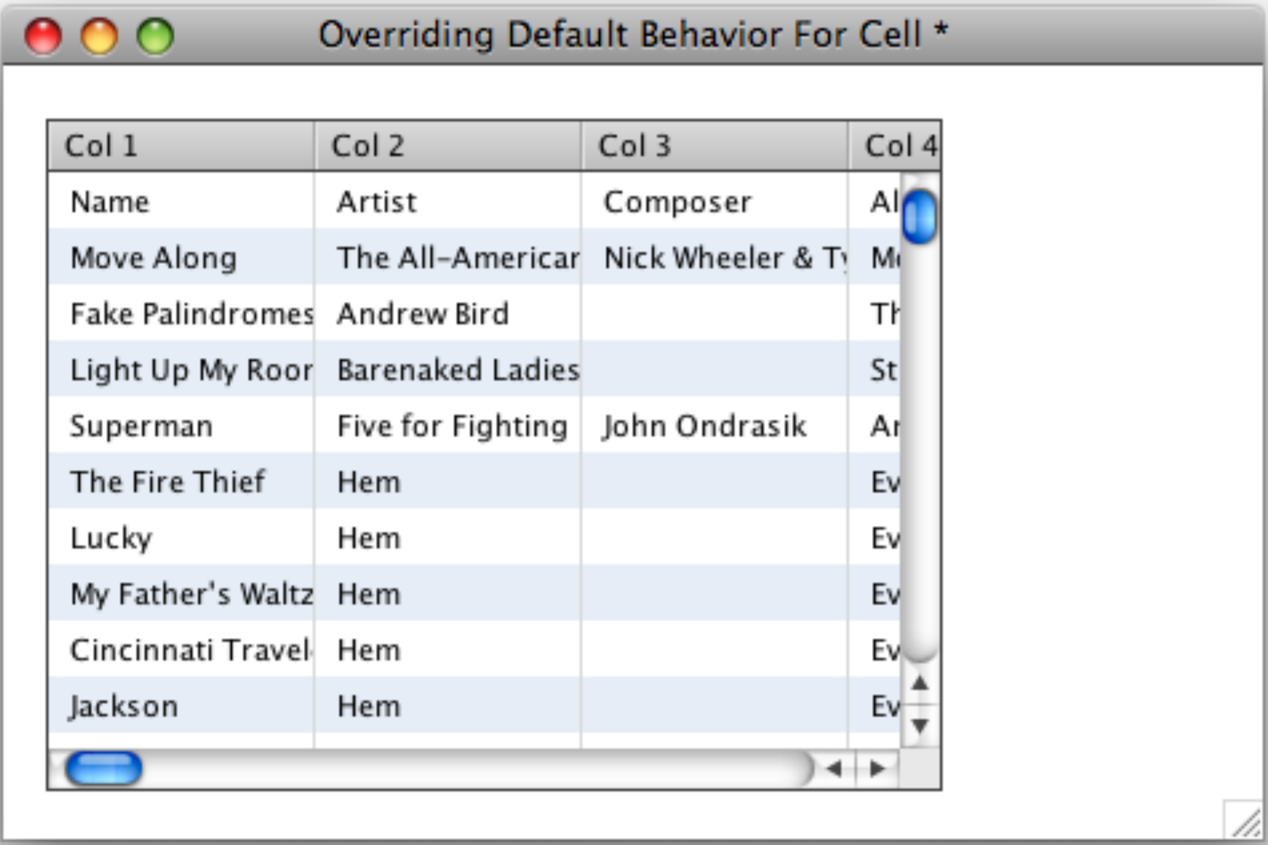
You can set the size of a column by setting the dgColumnWidth property for the column to an integer.

```
set the dgColumnWidth["Name"] of group "DataGrid" to 150
```

## How Do I Override the Default Behavior For Rendering Data to a Cell?

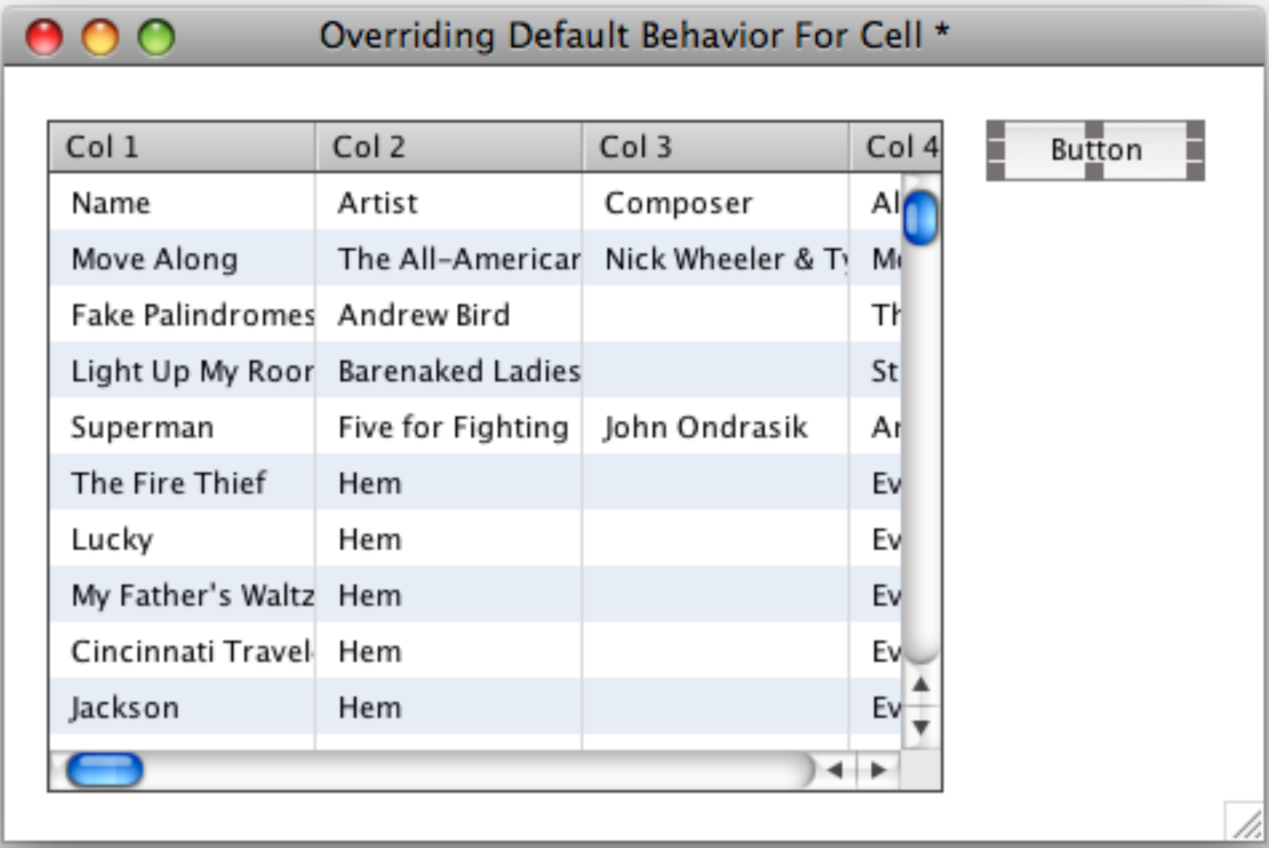
By default, a data grid table uses a single field object for each cell in a table and assigns the text property of that field to the cell's data. This lesson will show you how to quickly create your own script that determines how data is rendered in the default table cell. This can be useful for rendering HTML and unicode text, trailing off text that is too wide for a column or for coloring particular cells.

### Begin With a Data Grid Table



| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| Name | Artist | Composer | Al |
| Move Along | The All-Americar | Nick Wheeler & Ty | Mı |
| Fake Palindromes | Andrew Bird | | Th |
| Light Up My Roor | Barenaked Ladies | | St |
| Superman | Five for Fighting | John Ondrasik | Ar |
| The Fire Thief | Hem | | Ev |
| Lucky | Hem | | Ev |
| My Father's Waltz | Hem | | Ev |
| Cincinnati Travel | Hem | | Ev |
| Jackson | Hem | | Ev |

## Create a Button

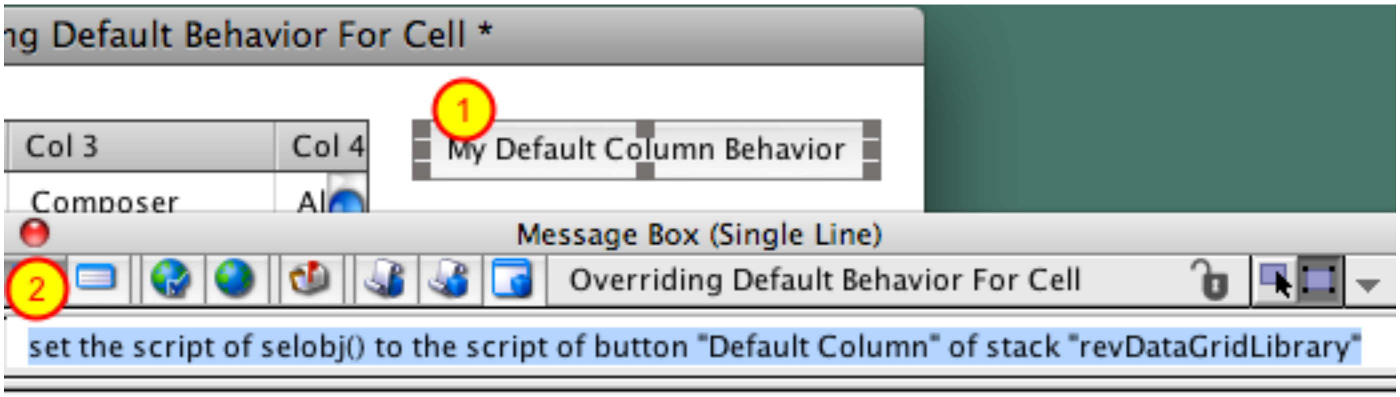


| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| Name | Artist | Composer | Al |
| Move Along | The All-Americar | Nick Wheeler & Ty | Mi |
| Fake Palindromes | Andrew Bird | | Th |
| Light Up My Roor | Barenaked Ladies | | St |
| Superman | Five for Fighting | John Ondrasik | Ar |
| The Fire Thief | Hem | | Ev |
| Lucky | Hem | | Ev |
| My Father's Waltz | Hem | | Ev |
| Cincinnati Travel | Hem | | Ev |
| Jackson | Hem | | Ev |

The **default column behavior** property can be set to a button. The script of the button will be used to fill in each cell in the table.

I'm going to rename the button to **My Default Column Behavior**.

## Set the Script Of The Button

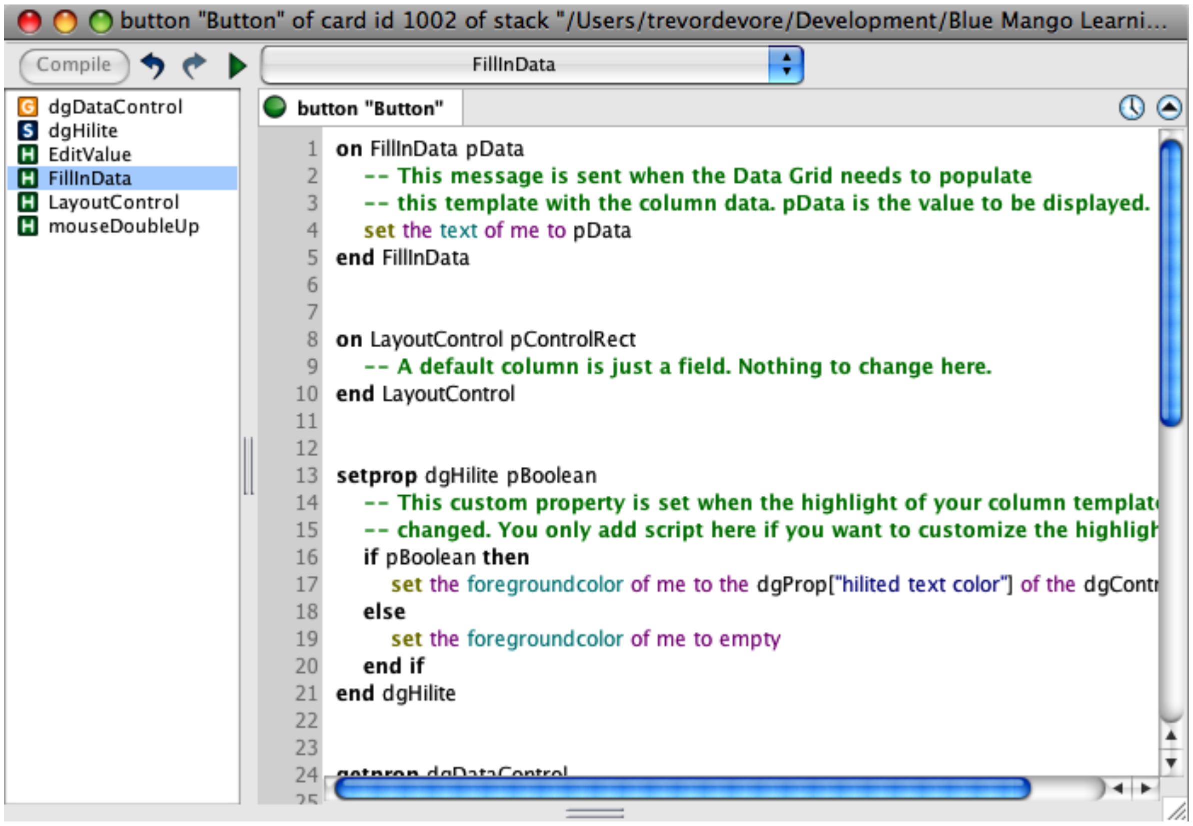

ng Default Behavior For Cell *
Col 3
Col 4
Composer
My Default Column Behavior
1
2
Message Box (Single Line)
Overriding Default Behavior For Cell

```
set the script of selobj() to the script of button "Default Column" of stack "revDataGridLibrary"
```

When creating your own **default column behavior** it is a good idea to start with the script that the data grid uses by default. You can copy the data grid default script easily enough by selecting the button (1) and executing the following statement in the Message Box (2):

set the script of selobj() to the script of button "Default Column" of stack "revDataGridLibrary"

## Customize Behavior

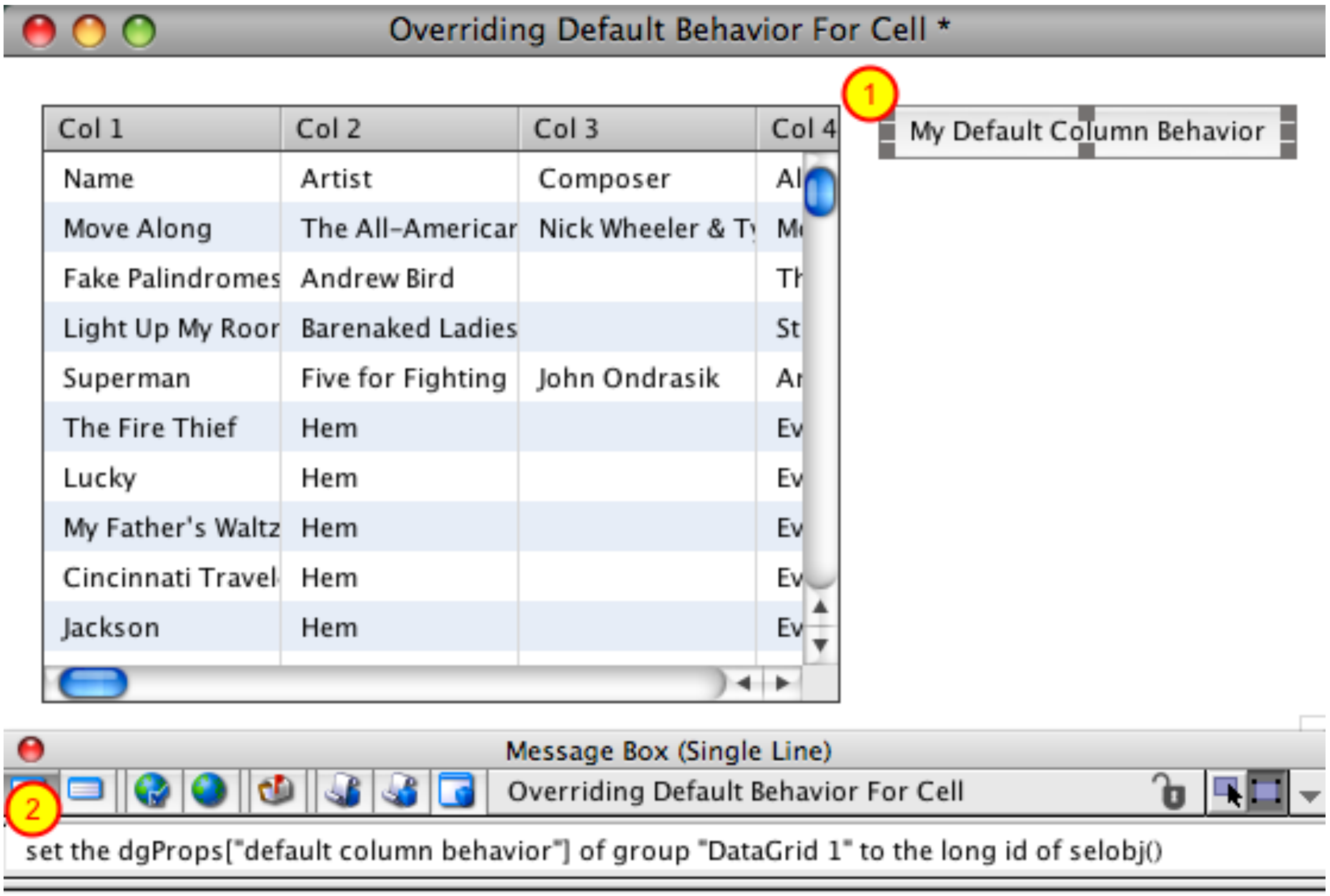


```
on FillInData pData
   -- This message is sent when the Data Grid needs to populate
   -- this template with the column data. pData is the value to be displayed.
   set the text of me to pData
end FillInData


on LayoutControl pControlRect
   -- A default column is just a field. Nothing to change here.
end LayoutControl


setprop dgHilite pBoolean
   -- This custom property is set when the highlight of your column templat
   -- changed. You only add script here if you want to customize the highligh
   if pBoolean then
      set the foregroundcolor of me to the dgProp["hilited text color"] of the dgContr
   else
      set the foregroundcolor of me to empty
   end if
end dgHilite


getprop dgDataControl
```

Now you can customize the behavior however you would like. Here is what the default behavior looks like.

**Important:** Make sure you include the dgDataControl getProp handler in your script. This is required in order for the data grid to work properly.

## Set 'default column behavior' Property


Overriding Default Behavior For Cell *
1
My Default Column Behavior
Message Box (Single Line)
2
Overriding Default Behavior For Cell

| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| Name | Artist | Composer | Al |
| Move Along | The All-Americar | Nick Wheeler & Ty | M |
| Fake Palindromes | Andrew Bird | | Th |
| Light Up My Roor | Barenaked Ladies | | St |
| Superman | Five for Fighting | John Ondrasik | Ar |
| The Fire Thief | Hem | | Ev |
| Lucky | Hem | | Ev |
| My Father's Waltz | Hem | | Ev |
| Cincinnati Travel | Hem | | Ev |
| Jackson | Hem | | Ev |

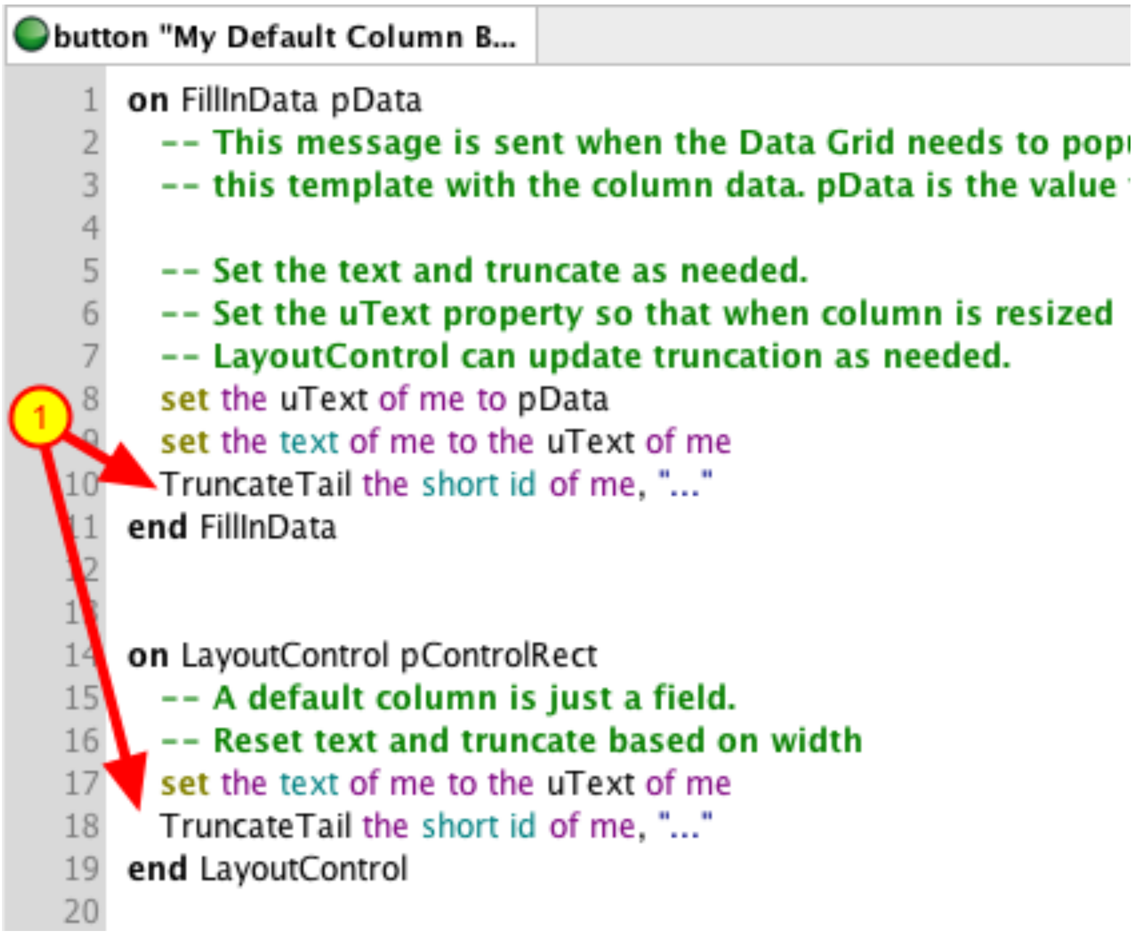
```
set the dgProps["default column behavior"] of group "DataGrid 1" to the long id of selobj()
```

Now you can set the **default column behavior** property of the data grid. Select the button (1) and execute the following in the Message Box (2):

set the dgProps["default column behavior"] of group "DataGrid 1" to the long id of selobj()

Now it is time to customize your script!

## Example: Truncate Tail



```
on FillInData pData
   -- This message is sent when the Data Grid needs to popu
   -- this template with the column data. pData is the value

   -- Set the text and truncate as needed.
   -- Set the uText property so that when column is resized
   -- LayoutControl can update truncation as needed.
   set the uText of me to pData
   set the text of me to the uText of me
   TruncateTail the short id of me, "..."
end FillInData


on LayoutControl pControlRect
   -- A default column is just a field.
   -- Reset text and truncate based on width
   set the text of me to the uText of me
   TruncateTail the short id of me, "..."
end LayoutControl
```

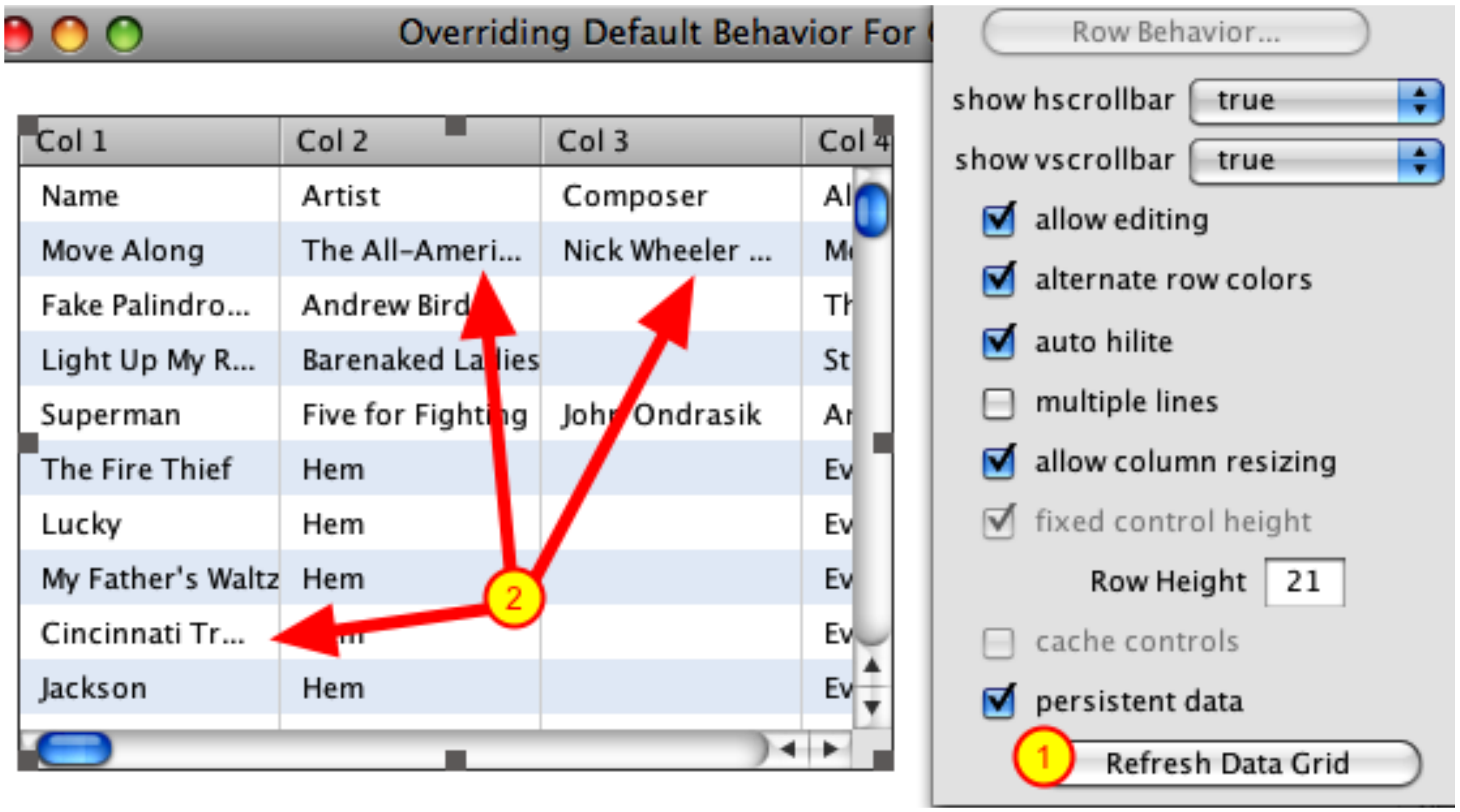
As an example, I will show you how to truncate the tail end of every cell whose content is too wide to fit. The data grid provides a helper command named **TruncateTail** that takes the short id of a field and a string that signifies the text is being truncated. I've added a call to **TruncateTail** to the **FillInData** and **LayoutControl** handlers so that cell contents are truncated when drawn or when a column is resized.

**Note:** TruncateTail works fairly well for most cases but can cause visual lag if there are lots of cells being displayed that use TruncateTail. You should test your data and table to make sure it performs adequately for your needs.

**Quick Tip:** You can determine the name of the column that is being rendered by checking the **dgColumn** property. You could use this property if you only wanted to truncate the text in certain columns:

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

## Refresh



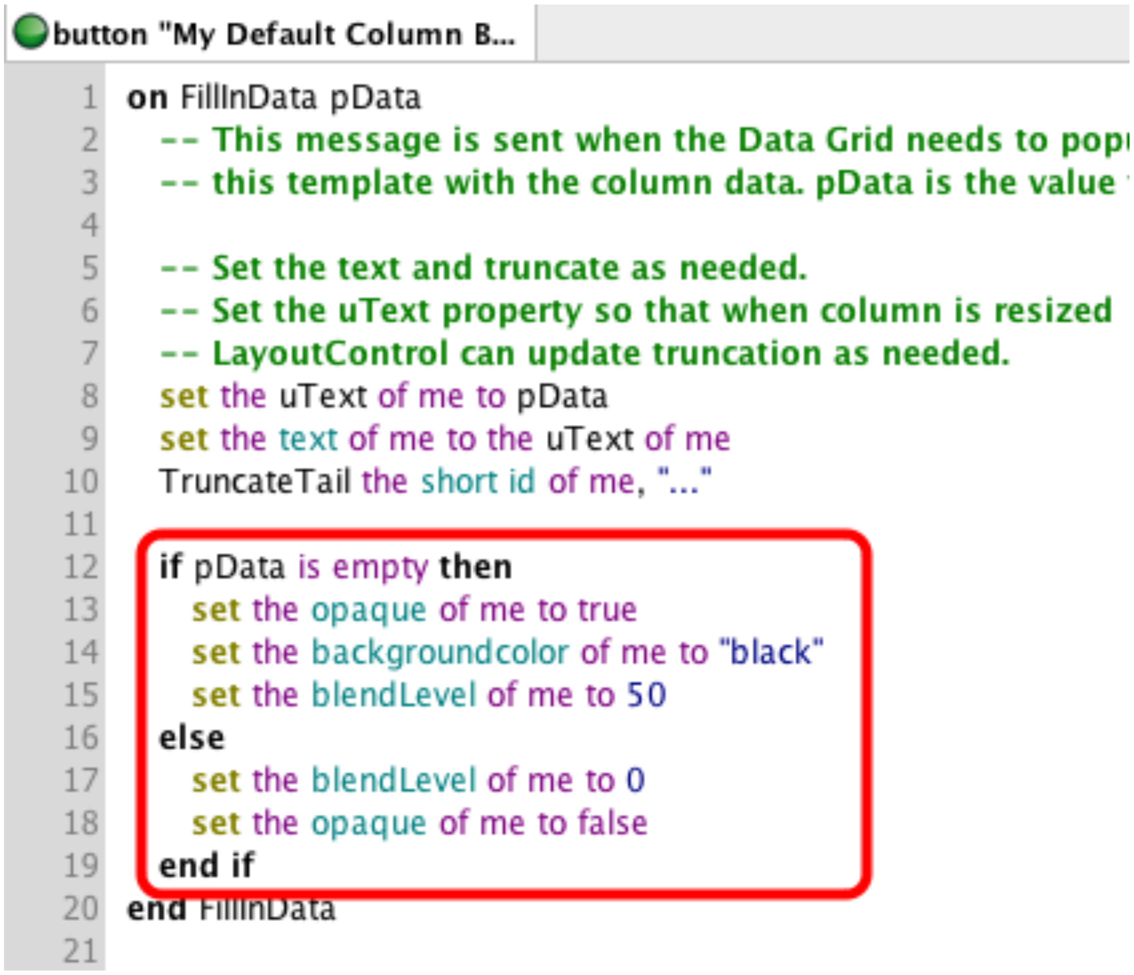
| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| Name | Artist | Composer | Al |
| Move Along | The All-Ameri... | Nick Wheeler ... | Mo |
| Fake Palindro... | Andrew Bird | | Th |
| Light Up My R... | Barenaked Ladies | | St |
| Superman | Five for Fighting | John Ondrasik | Ar |
| The Fire Thief | Hem | | Ev |
| Lucky | Hem | | Ev |
| My Father's Waltz | Hem | | Ev |
| Cincinnati Tr... | [illegible] | | Ev |
| Jackson | Hem | | Ev |

show hscrollbar: true
show vscrollbar: true
- [x] allow editing
- [x] alternate row colors
- [x] auto hilite
- [ ] multiple lines
- [x] allow column resizing
- [x] fixed control height

Row Height: 21
- [ ] cache controls
- [x] persistent data

To see the results, click the **Refresh Data Grid** button in the Property Inspector (1). Notice how cell contents are no truncated as needed (2).

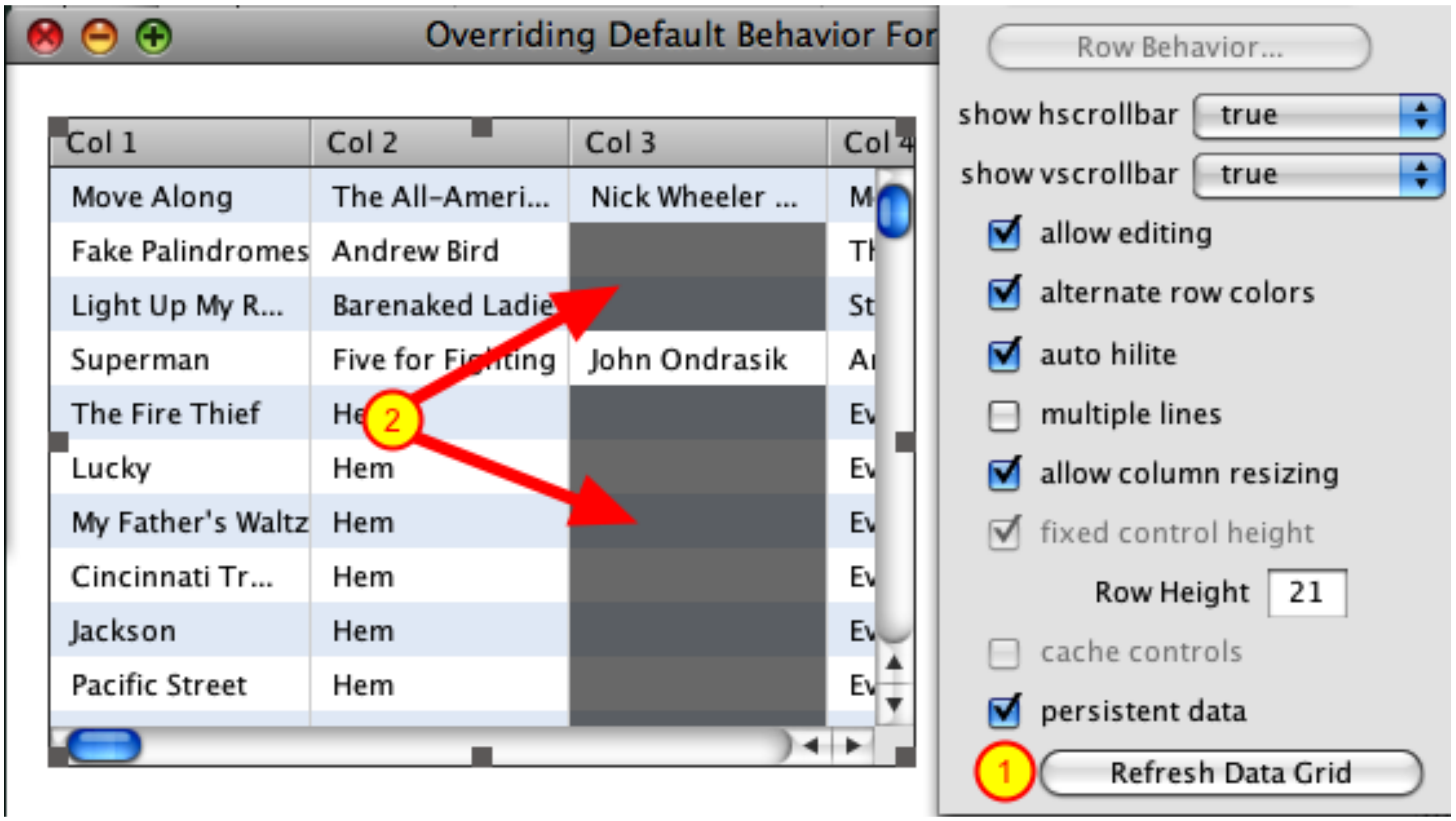
## Example: Coloring Cells



```
on FillInData pData
   -- This message is sent when the Data Grid needs to popu
   -- this template with the column data. pData is the value

   -- Set the text and truncate as needed.
   -- Set the uText property so that when column is resized
   -- LayoutControl can update truncation as needed.
   set the uText of me to pData
   set the text of me to the uText of me
   TruncateTail the short id of me, "..."

   if pData is empty then
      set the opaque of me to true
      set the backgroundcolor of me to "black"
      set the blendLevel of me to 50
   else
      set the blendLevel of me to 0
      set the opaque of me to false
   end if
end FillInData
```

In this example I am going to dim any cell that is empty. Since the default cell consists of a single field object I can just set the opaque to true, set the backgroundcolor and change the blendlevel.

## Result



| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| Move Along | The All-Ameri... | Nick Wheeler ... | M |
| Fake Palindromes | Andrew Bird | | Th |
| Light Up My R... | Barenaked Ladie | | St |
| Superman | Five for Fighting | John Ondrasik | A |
| The Fire Thief | He | | Ev |
| Lucky | Hem | | Ev |
| My Father's Waltz | Hem | | Ev |
| Cincinnati Tr... | Hem | | Ev |
| Jackson | Hem | | Ev |
| Pacific Street | Hem | | Ev |

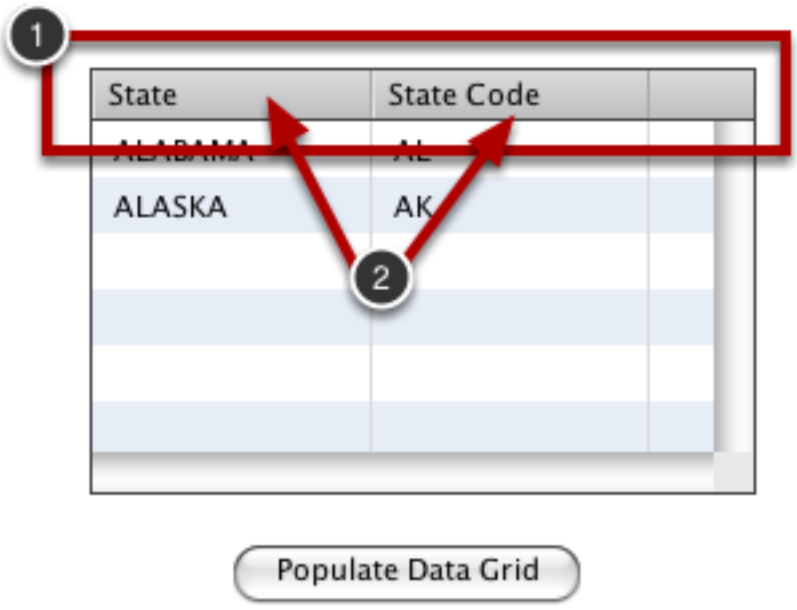
show hscrollbar: true
show vscrollbar: true
✓ allow editing
✓ alternate row colors
✓ auto hilite
☐ multiple lines
✓ allow column resizing
✓ fixed control height
Row Height: 21
☐ cache controls
✓ persistent data

By clicking on the **Refresh Data Grid** button (1) I can see the result (2).

## How Do I Determine If a User Clicks In The Table Header?

This lesson will show you how to determine if the user clicked in the header of a Data Grid table by showing you how to use the dgHeaderControl and dgHeader properties of the target control that was clicked on.

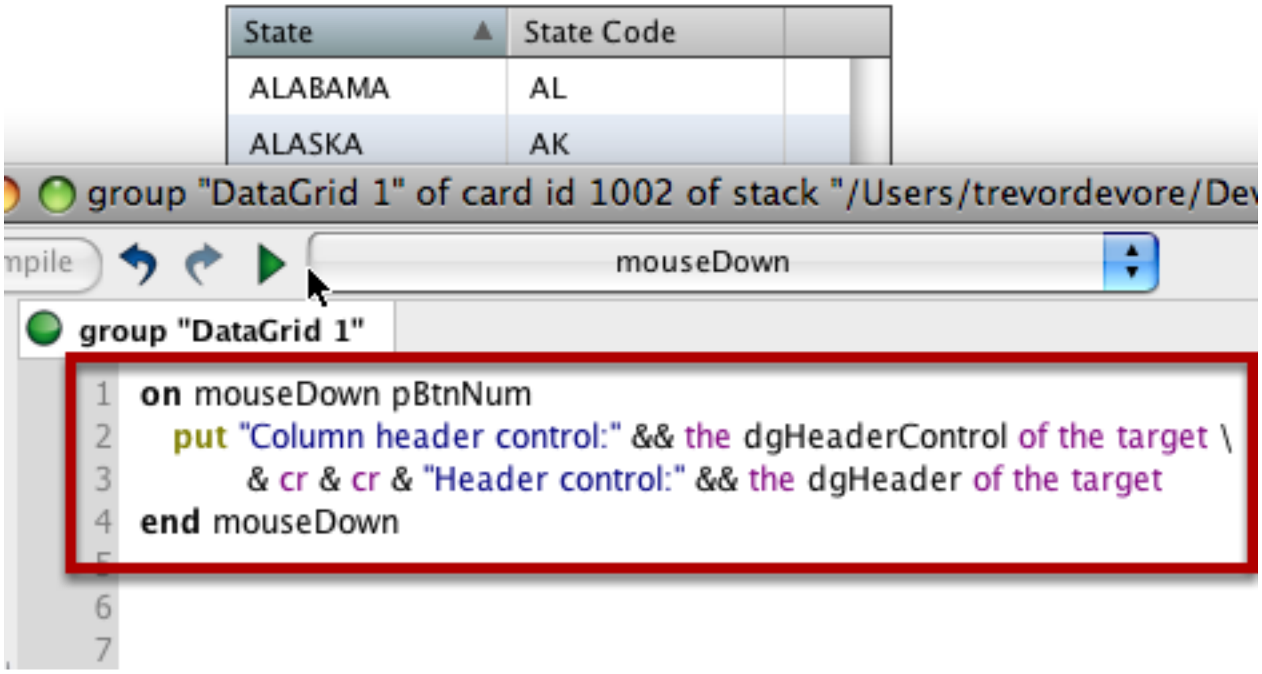
### The Table Header



A Data Grid table has a header where the column names are displayed (1). When a user clicks in the header with the mouse **the dgHeader of the target** returns the long id of the group control that contains all of the header controls.

When the user clicks on a column header (2) **the dgHeaderControl of the target** returns the long id of the group control that contains all of the column header controls.

## Example

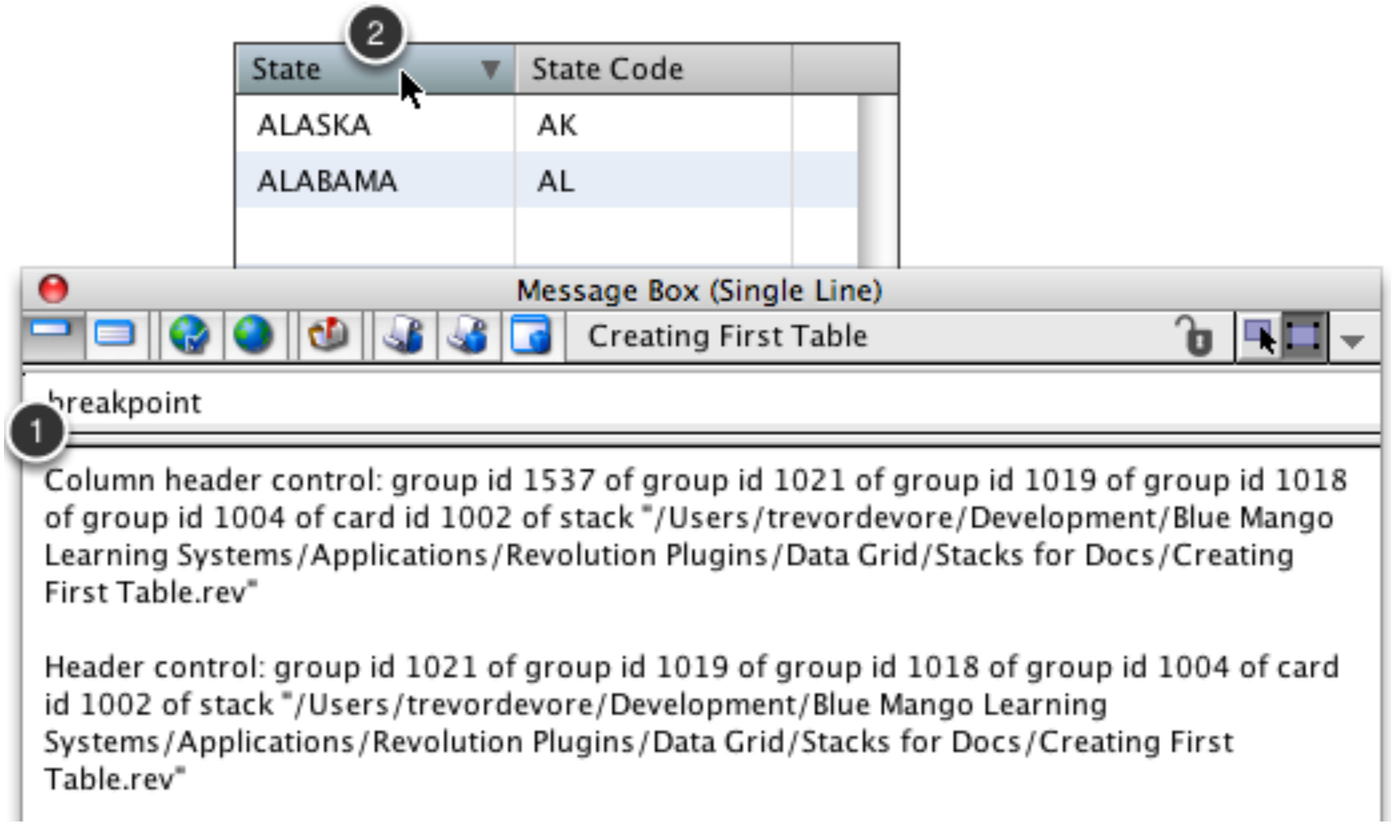

State
State Code
ALABAMA
AL
ALASKA
AK
group "DataGrid 1" of card id 1002 of stack "/Users/trevordevore/De
mpile
mouseDown
group "DataGrid 1"

```
on mouseDown pBtnNum
   put "Column header control:" && the dgHeaderControl of the target \
         & cr & cr & "Header control:" && the dgHeader of the target
end mouseDown
```

Here is some sample code that prints off the dgHeaderControl and dgHeader properties when the user clicks on a Data Grid. You can place this script in the Data Grid script.

**==========**
***Copy & Paste***
**==========**

```
on mouseDown pBtnNum
   put "Column header control:" && the dgHeaderControl of the target \
         & cr & cr & "Header control:" && the dgHeader of the target
end mouseDown
```

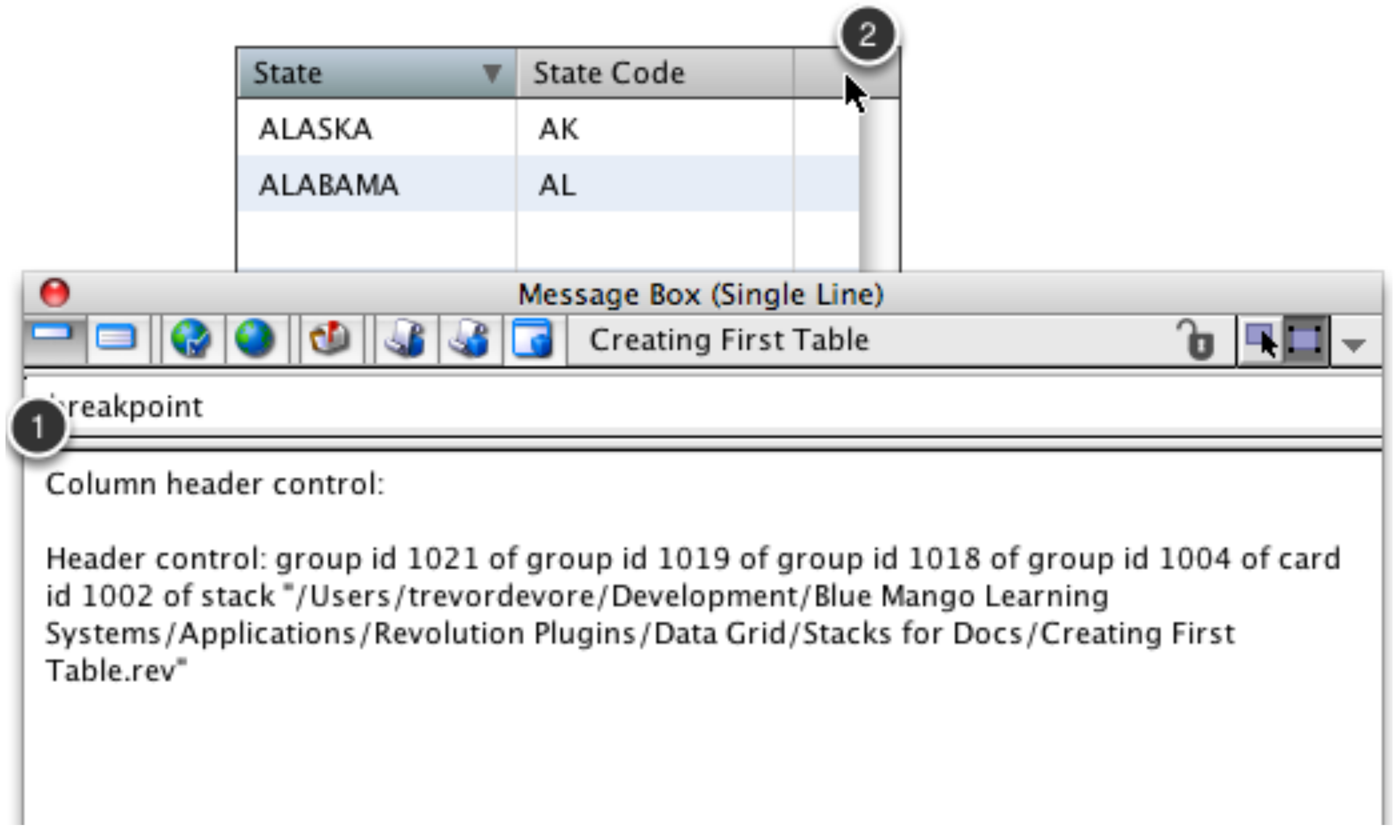


| State | State Code |
|---|---|
| ALASKA | AK |
| ALABAMA | AL |

```
breakpoint
```

```
Column header control: group id 1537 of group id 1021 of group id 1019 of group id 1018
of group id 1004 of card id 1002 of stack "/Users/trevordevore/Development/Blue Mango
Learning Systems/Applications/Revolution Plugins/Data Grid/Stacks for Docs/Creating
First Table.rev"

Header control: group id 1021 of group id 1019 of group id 1018 of group id 1004 of card
id 1002 of stack "/Users/trevordevore/Development/Blue Mango Learning
Systems/Applications/Revolution Plugins/Data Grid/Stacks for Docs/Creating First
Table.rev"
```

After adding the script to the Data Grid the value of the dgHeaderControl and dgHeader properties are displayed in the message box. In this example both properties return the long id of a control (1) because I clicked on a column header (2).

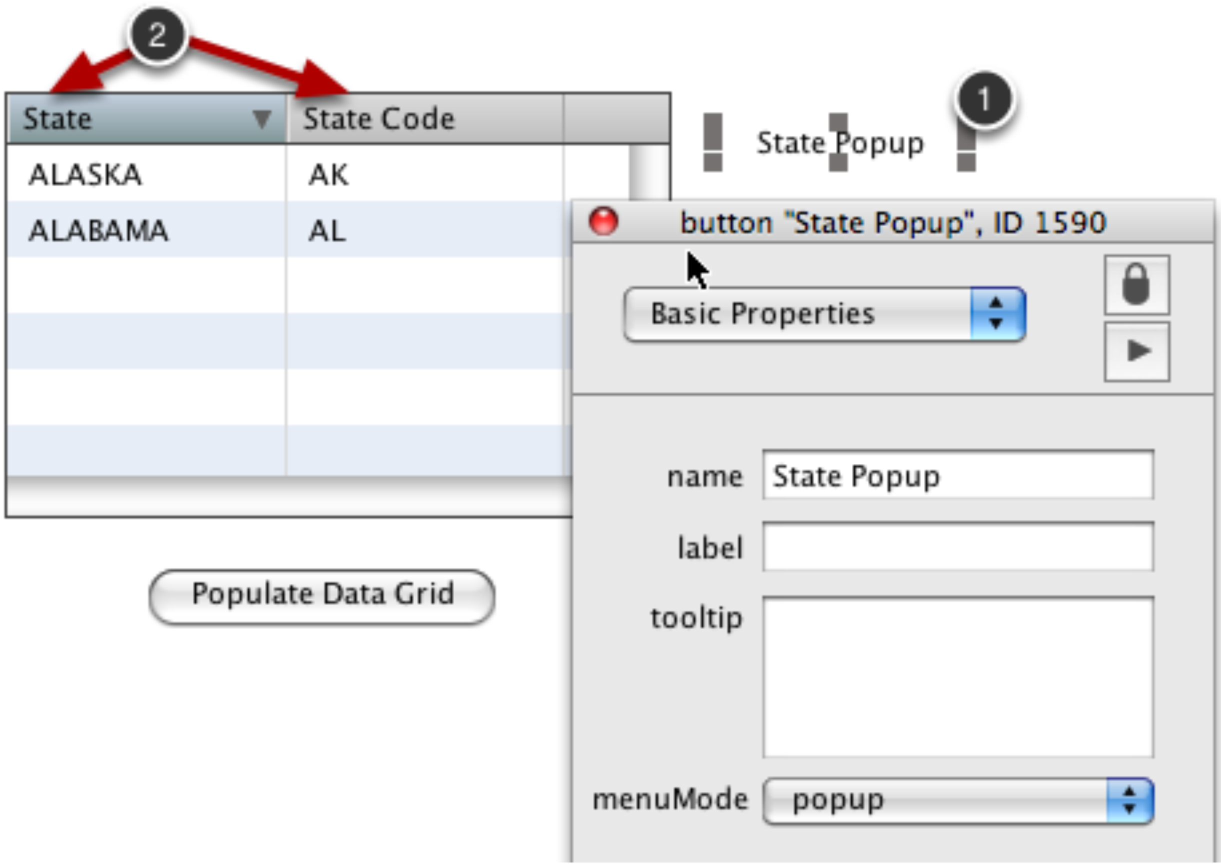


```
Column header control:

Header control: group id 1021 of group id 1019 of group id 1018 of group id 1004 of card
id 1002 of stack "/Users/trevordevore/Development/Blue Mango Learning
Systems/Applications/Revolution Plugins/Data Grid/Stacks for Docs/Creating First
Table.rev"
```

In this example only the dgHeader property returns the long id of a control (1) because I clicked in the table header but not on a column header (2).

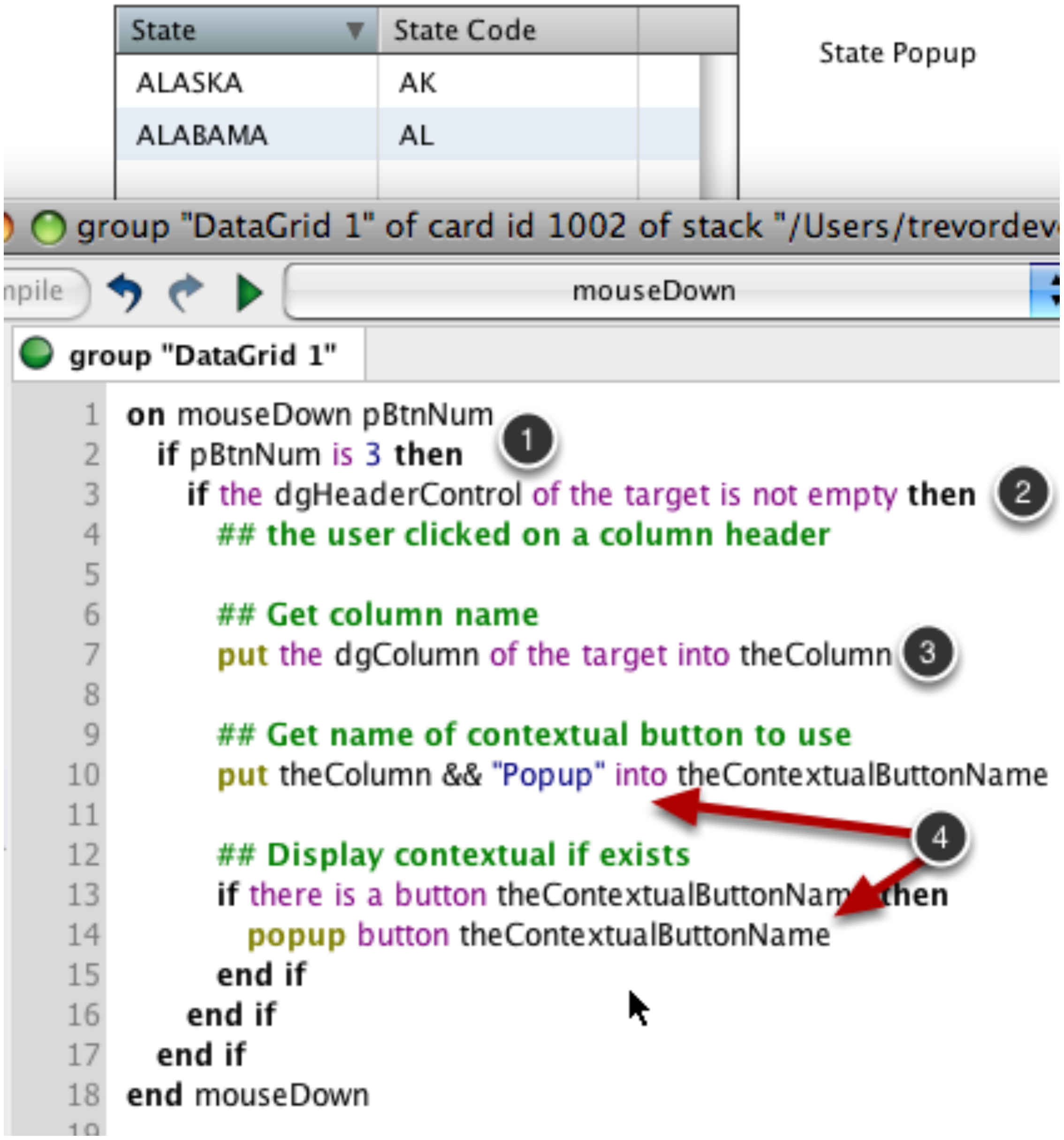
## How Do I Display a Contextual Menu When the User Clicks on a Column Header?

### Overview



This lesson will show you how to display a contextual menu (1) when the user clicks on a column header in a Data Grid table (2).

## The Code



```
on mouseDown pBtnNum
  if pBtnNum is 3 then
    if the dgHeaderControl of the target is not empty then
      ## the user clicked on a column header

      ## Get column name
      put the dgColumn of the target into theColumn

      ## Get name of contextual button to use
      put theColumn && "Popup" into theContextualButtonName

      ## Display contextual if exists
      if there is a button theContextualButtonName then
        popup button theContextualButtonName
      end if
    end if
  end if
end mouseDown
```

Let's look at some RevTalk code that will display a contextual menu. The code is located in the Data Grid group script.

A contextual menu is displayed when during the mouseDown message when the user clicks with the right mouse button (1).

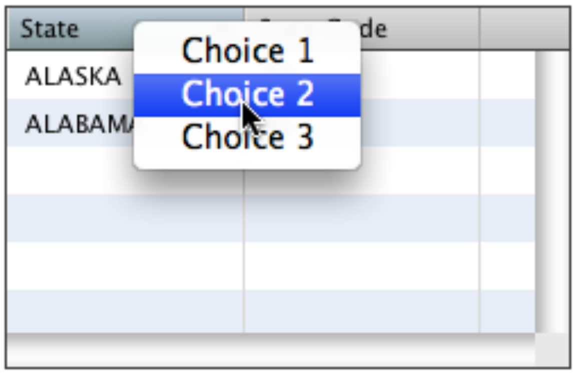
To determine if the user clicked on a column header you can check whether or not the dgHeaderControl property of the control that was clicked on (the target) is empty (2). If the property is not empty then you know that the user clicked on a column header.

Once you know that the user clicked on a column header you can display the contextual menu. In this example I take the name of the column that was clicked on (3) and append " Popup" to it in order to get the name of the popup button to display (4).

**==========**
***Copy & Paste***
**==========**

```
on mouseDown pBtnNum
   ## Let Data Grid process mouseDown
   dgMouseDown pBtnNum

   if pBtnNum is 3 then
      if the dgHeaderControl of the target is not empty then
         ## the user clicked on a column header

         ## Get column name
         put the dgColumn of the target into theColumn

         ## Get name of contextual button to use
         put theColumn && "Popup" into theContextualButtonName

         ## Display contextual if exists
         if there is a button theContextualButtonName then
            popup button theContextualButtonName
         end if
      end if
   end if
end mouseDown
```

## The Result



Now when I right-click on the State column the **State Popup** popup button is displayed as a contextual menu.

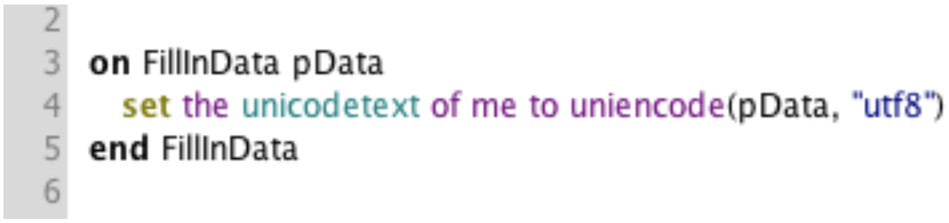
## What If I Need to Work With htmlText, rtfText or unicodeText?

By default all table columns will have their text property assigned. If you need to work with unicode, htmltext or rtftext then you can provide your own behavior for table columns.

### Option 1: Set the dgProp["default column behavior"] Property

The simplest way to change the default behavior of table columns is to set the dgProp["default column behavior"] property of the data grid. You set this property to a button id whose script contains the behavior you would like the columns to have. Doing this overrides the default behavior provided by a button on the revDataGridLibrary stack.

```
set the dgProp["default column behavior"] of group "DataGrid" to the long id of button "My Custom Column Behavior" of stack "MyStack"
```



```

on FillInData pData
   set the unicodetext of me to uniencode(pData, "utf8")
end FillInData

```

Here is an example of a FillInData command that takes a column value encoded as UTF-8 and sets the unicodetext of the column field.

To see the default column script the data grid uses enter the following in the message box:

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

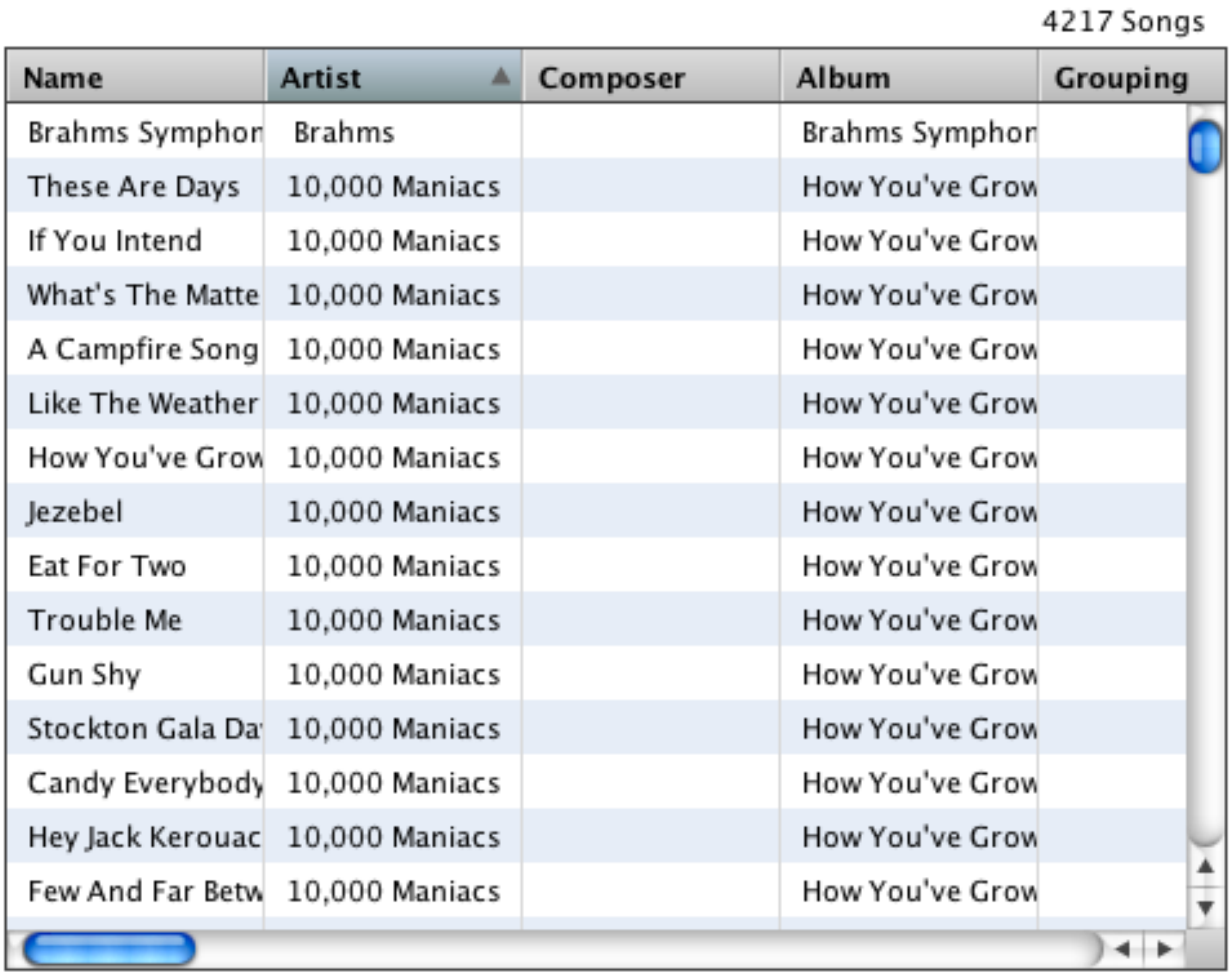
### Option 2: Create a Custom Column Template

Alternatively you can create your own custom column template for a particular column. This allows you complete contorl over the look and feel of the column. See the lesson on creating custom column templates.

## How Do I Display Line Numbers in a Table?

This lesson will show you how to dynamically display line numbers in your data grid table.

### The Table

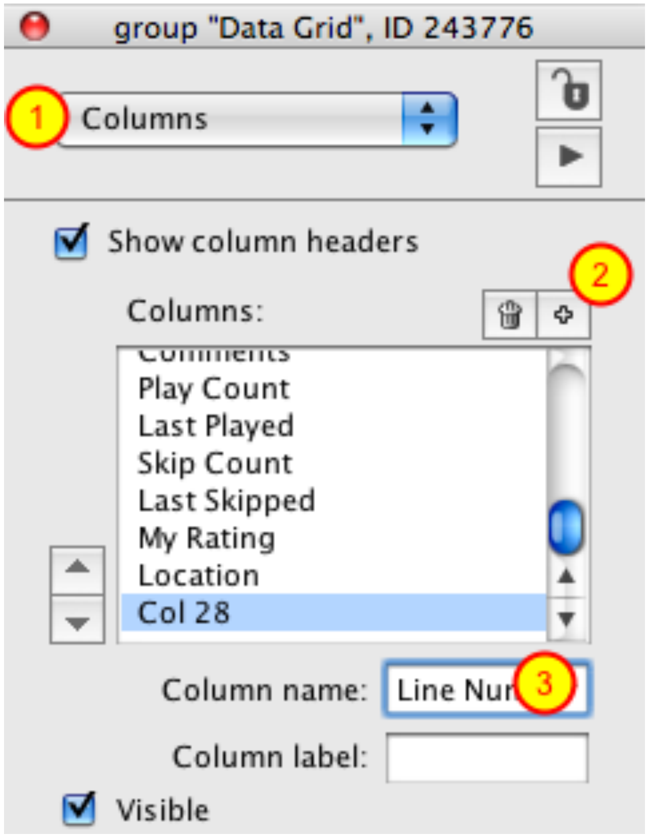


4217 Songs

| Name | Artist | Composer | Album | Grouping |
|---|---|---|---|---|
| Brahms Symphon | Brahms | | Brahms Symphon | |
| These Are Days | 10,000 Maniacs | | How You've Grow | |
| If You Intend | 10,000 Maniacs | | How You've Grow | |
| What's The Matte | 10,000 Maniacs | | How You've Grow | |
| A Campfire Song | 10,000 Maniacs | | How You've Grow | |
| Like The Weather | 10,000 Maniacs | | How You've Grow | |
| How You've Grow | 10,000 Maniacs | | How You've Grow | |
| Jezebel | 10,000 Maniacs | | How You've Grow | |
| Eat For Two | 10,000 Maniacs | | How You've Grow | |
| Trouble Me | 10,000 Maniacs | | How You've Grow | |
| Gun Shy | 10,000 Maniacs | | How You've Grow | |
| Stockton Gala Da | 10,000 Maniacs | | How You've Grow | |
| Candy Everybody | 10,000 Maniacs | | How You've Grow | |
| Hey Jack Kerouac | 10,000 Maniacs | | How You've Grow | |
| Few And Far Betw | 10,000 Maniacs | | How You've Grow | |

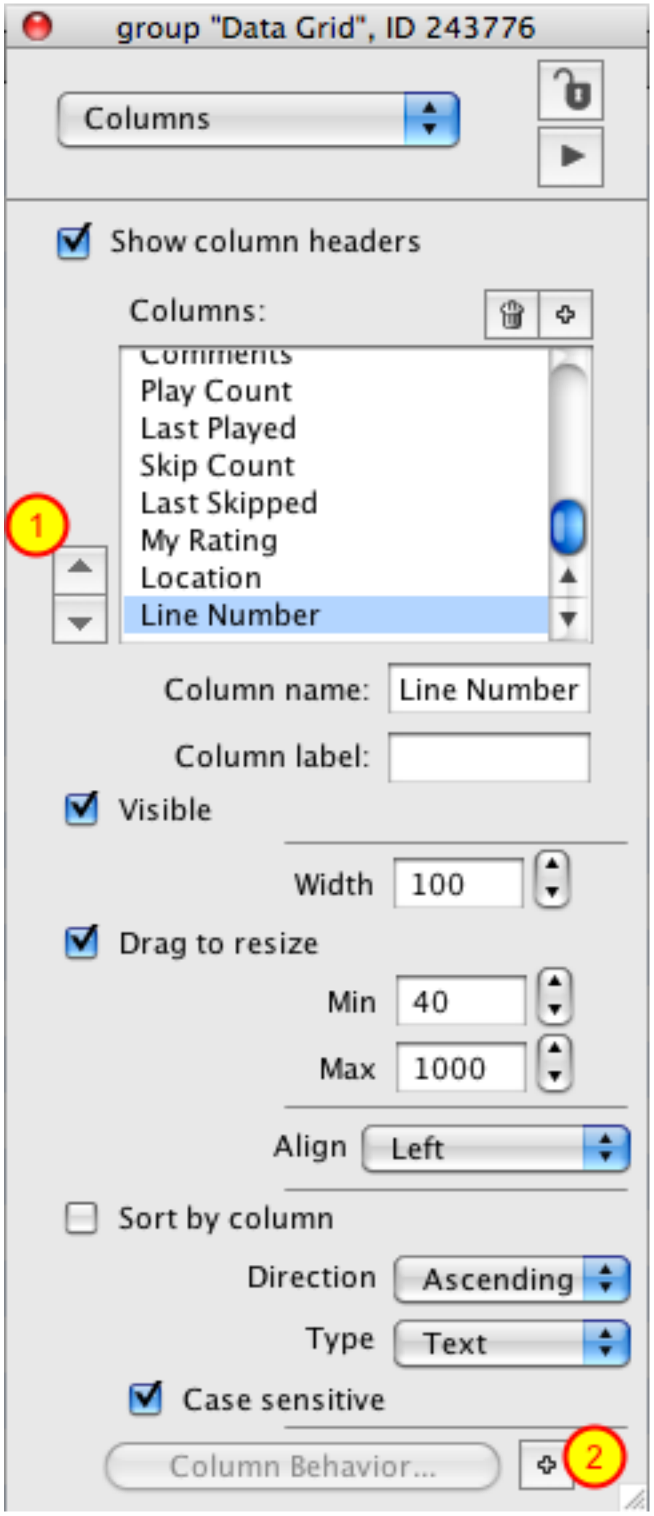
Here is the table that I will display line numbers in.

## Add Line Number Column



Select the data grid and open the property inspector. Select the Columns pane (1) and add a column using the "+" button (2). Rename the column "Line Number" (3).

## Reorder and Create Column Template


group "Data Grid", ID 243776

Columns

Show column headers ✓

Columns:

Comments
Play Count
Last Played
Skip Count
Last Skipped
My Rating
Location
Line Number

Column name: Line Number

Column label:

Visible ✓

Width 100

Drag to resize ✓

Min 40

Max 1000

Align Left

Sort by column

Direction Ascending

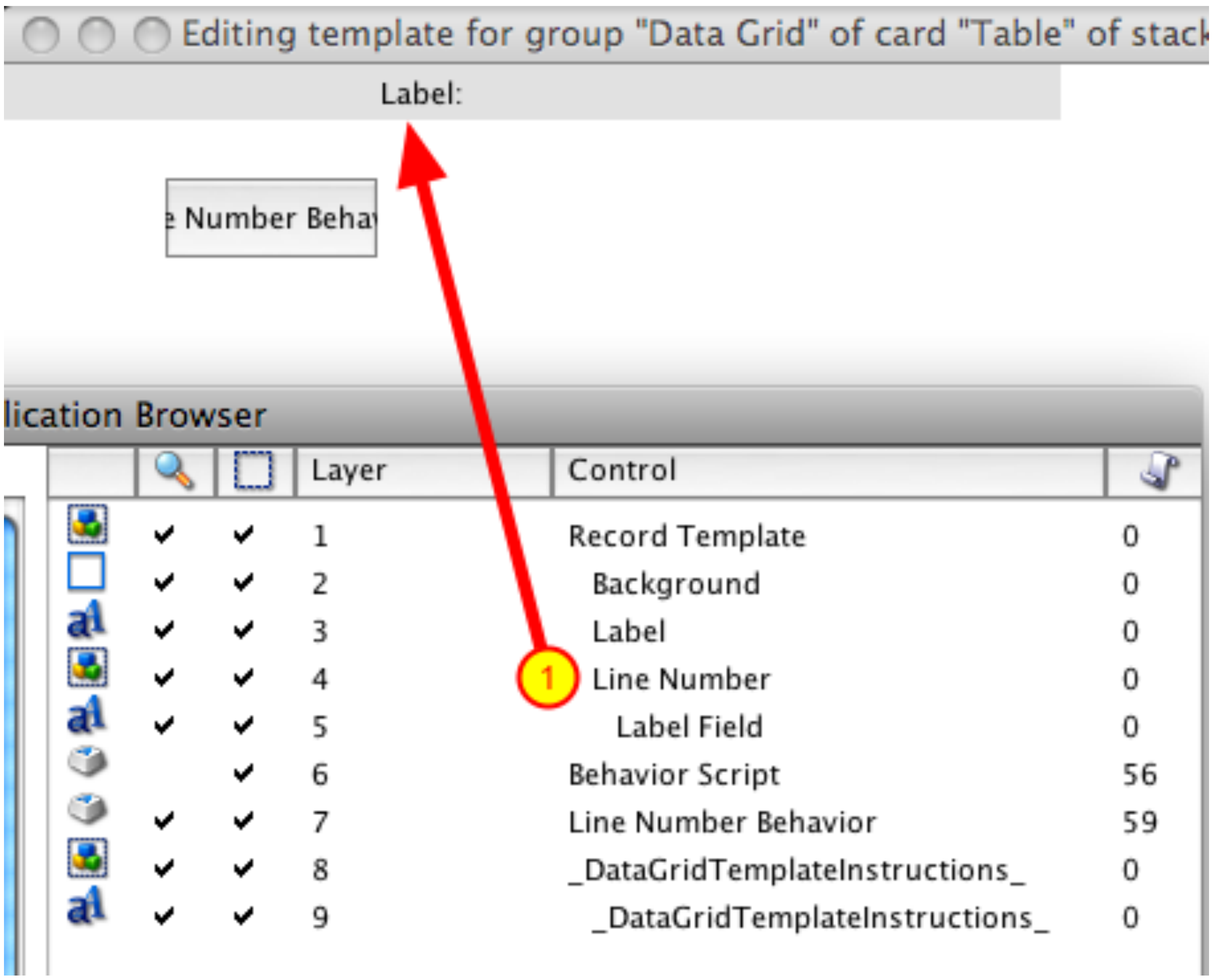
Type Text

Case sensitive ✓

Use the up arrow (1) to move the Line Number column to the top of the list.

Create a template for this column by clicking on the "+" button at the bottom of the property inspector (2).

## Edit Line Number Group



Editing template for group "Data Grid" of card "Table" of stack

Label:

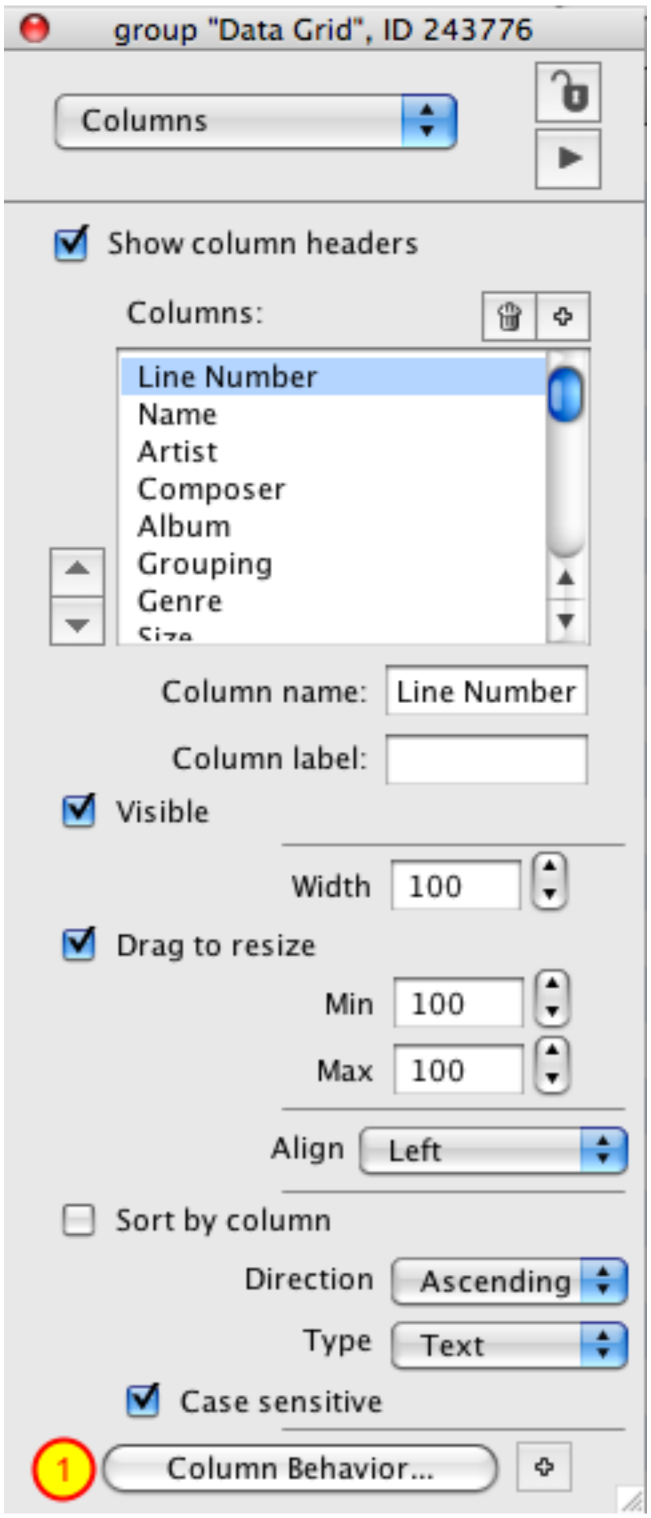
Number Beha

lication Browser

| | | | Layer | Control | |
|---|---|---|---|---|---|
| | ✓ | ✓ | 1 | Record Template | 0 |
| | ✓ | ✓ | 2 | Background | 0 |
| | ✓ | ✓ | 3 | Label | 0 |
| | ✓ | ✓ | 4 | Line Number | 0 |
| | ✓ | ✓ | 5 | Label Field | 0 |
| | | ✓ | 6 | Behavior Script | 56 |
| | ✓ | ✓ | 7 | Line Number Behavior | 59 |
| | ✓ | ✓ | 8 | _DataGridTemplateInstructions_ | 0 |
| | ✓ | ✓ | 9 | _DataGridTemplateInstructions_ | 0 |

After creating the column template the card with the template group will open.

## Edit Column Behavior



group "Data Grid", ID 243776

Columns

[✓] Show column headers

Columns:

Line Number
Name
Artist
Composer
Album
Grouping
Genre
Size

Column name: Line Number

Column label:

[✓] Visible

Width 100

[✓] Drag to resize

Min 100

Max 100

Align Left

[ ] Sort by column

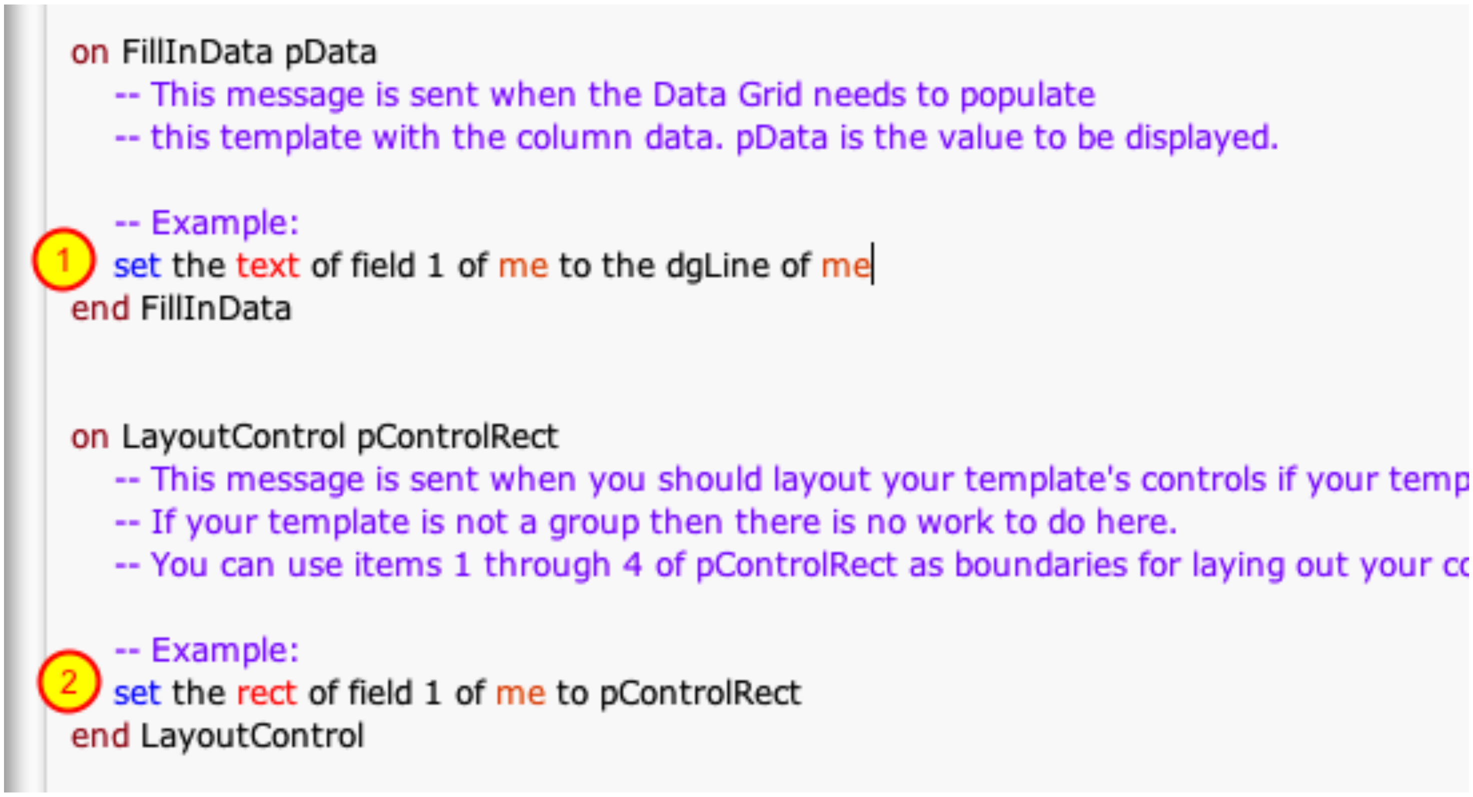
Direction Ascending

Type Text

[✓] Case sensitive

Now all that is left is to define the behavior for this column. Back in the Columns pane click the **Column Behavior** button (1) to open the script editor.

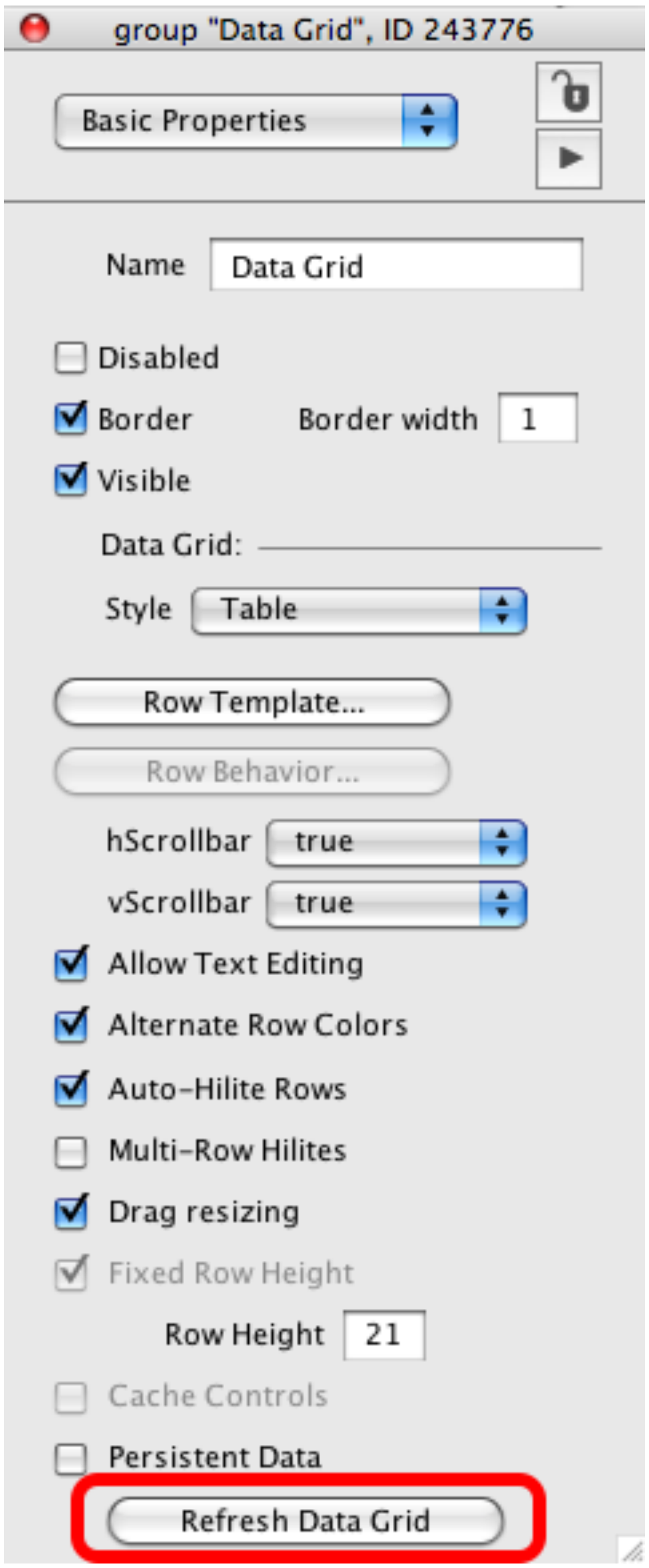
## Edit Behavior Script



```
on FillInData pData
   -- This message is sent when the Data Grid needs to populate
   -- this template with the column data. pData is the value to be displayed.

   -- Example:
1  set the text of field 1 of me to the dgLine of me
end FillInData


on LayoutControl pControlRect
   -- This message is sent when you should layout your template's controls if your temp
   -- If your template is not a group then there is no work to do here.
   -- You can use items 1 through 4 of pControlRect as boundaries for laying out your co

   -- Example:
2  set the rect of field 1 of me to pControlRect
end LayoutControl
```

There isn't much to do here. In FillInData you just want to 'set the text of field 1 of me to the dgLine of me' (1). The dgLine is a property of a row template and this will display the line number for the current line. In LayoutControl you just need to set the rect of the field so that it fills the entire cell (2).

## Refresh Data Grid


group "Data Grid", ID 243776

Basic Properties

Name: Data Grid

- [ ] Disabled
- [x] Border — Border width: 1
- [x] Visible

Data Grid:

Style: Table

hScrollbar: true

vScrollbar: true

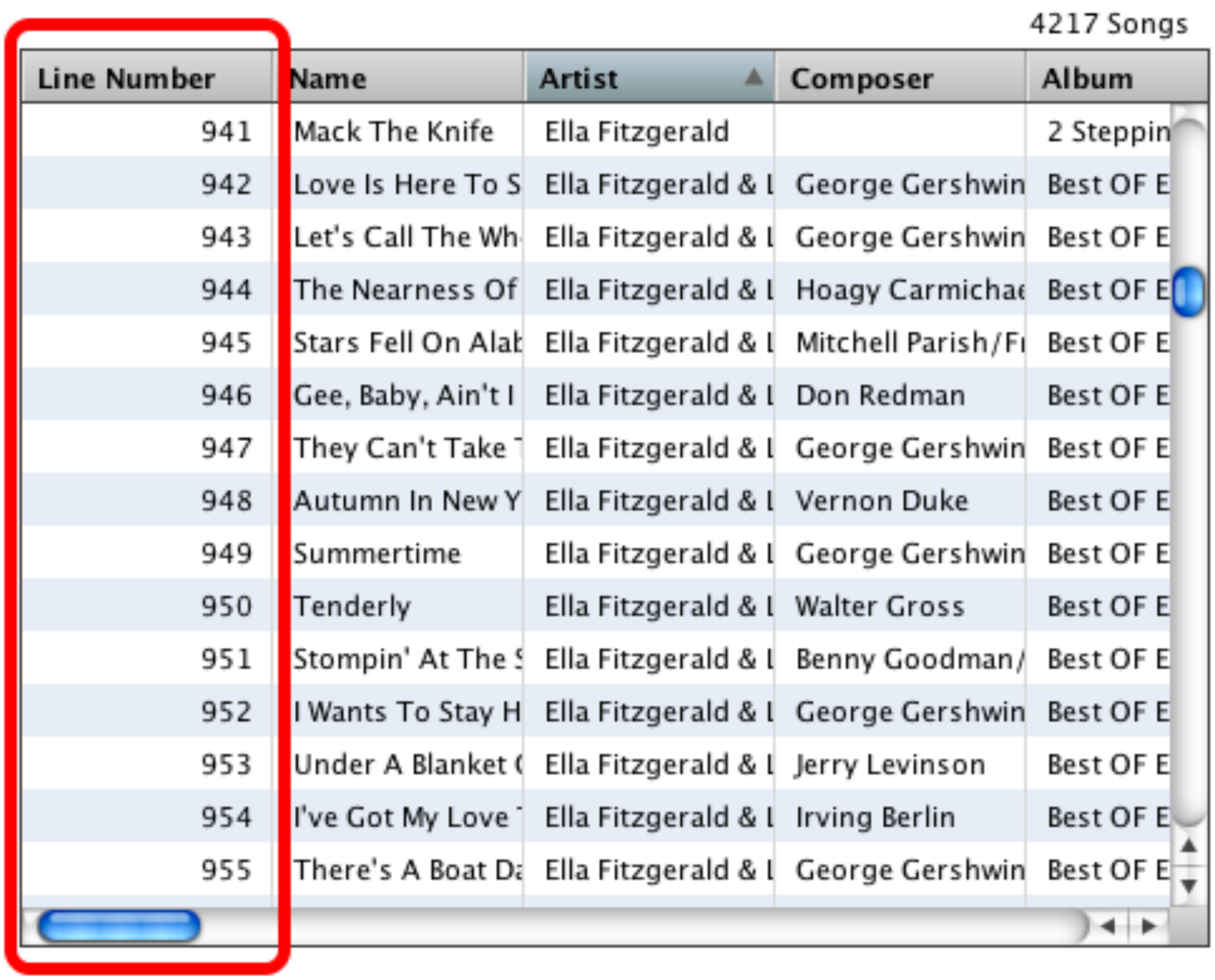
- [x] Allow Text Editing
- [x] Alternate Row Colors
- [x] Auto-Hilite Rows
- [ ] Multi-Row Hilites
- [x] Drag resizing
- [x] Fixed Row Height

Row Height: 21

- [ ] Cache Controls
- [ ] Persistent Data

In the property inspector click the **Refresh Data Grid** button.

## The Result



4217 Songs

| Line Number | Name | Artist | Composer | Album |
|---|---|---|---|---|
| 941 | Mack The Knife | Ella Fitzgerald | | 2 Steppin |
| 942 | Love Is Here To S | Ella Fitzgerald & L | George Gershwin | Best OF E |
| 943 | Let's Call The Wh | Ella Fitzgerald & L | George Gershwin | Best OF E |
| 944 | The Nearness Of | Ella Fitzgerald & L | Hoagy Carmichae | Best OF E |
| 945 | Stars Fell On Alal | Ella Fitzgerald & L | Mitchell Parish/Fi | Best OF E |
| 946 | Gee, Baby, Ain't I | Ella Fitzgerald & L | Don Redman | Best OF E |
| 947 | They Can't Take | Ella Fitzgerald & L | George Gershwin | Best OF E |
| 948 | Autumn In New Y | Ella Fitzgerald & L | Vernon Duke | Best OF E |
| 949 | Summertime | Ella Fitzgerald & L | George Gershwin | Best OF E |
| 950 | Tenderly | Ella Fitzgerald & L | Walter Gross | Best OF E |
| 951 | Stompin' At The S | Ella Fitzgerald & L | Benny Goodman/ | Best OF E |
| 952 | I Wants To Stay H | Ella Fitzgerald & L | George Gershwin | Best OF E |
| 953 | Under A Blanket ( | Ella Fitzgerald & L | Jerry Levinson | Best OF E |
| 954 | I've Got My Love | Ella Fitzgerald & L | Irving Berlin | Best OF E |
| 955 | There's A Boat D | Ella Fitzgerald & L | George Gershwin | Best OF E |

Now the line numbers appear in the table.

## But How Do I Set the dgText Then?

You may be wondering how you would set the dgText property of this data grid since we have added a "Line Number" column as the first column in the table.

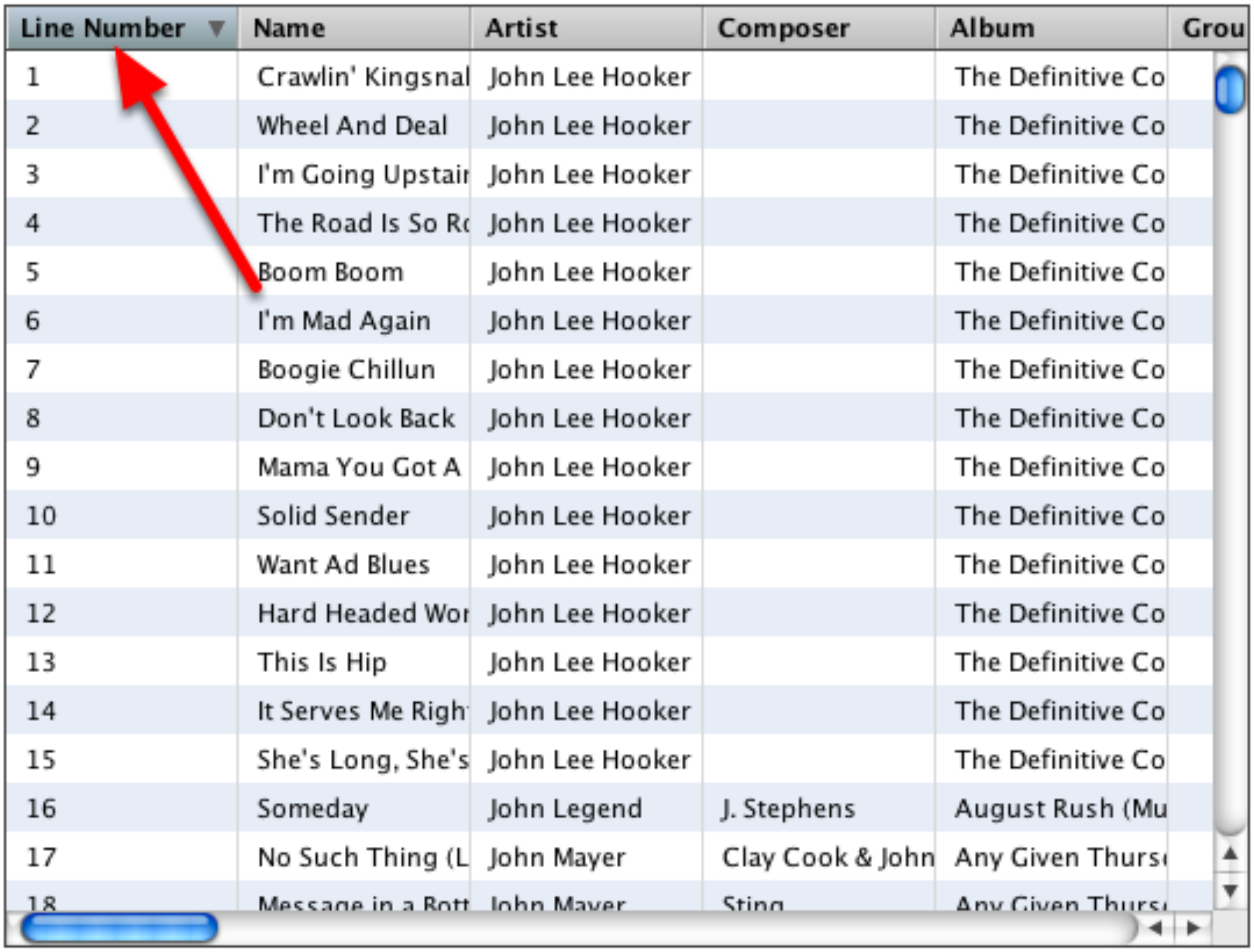
When you set the dgText of a data grid you can pass a parameter specifying whether or not the first line of text contains the names of the columns that the data should map to. Just pass in true and prepend the column names to your data like this:

```
put "Name,Artist,Composer,Album" into theColNames
set the dgText [true] of group "Data Grid" to theColNames & cr & theData
```

## How Do I Customize Column Sorting?

This lesson will demonstrate how to customize sorting for columns by handling the **SortDataGridColumn** message that is sent when the user clicks on a column.

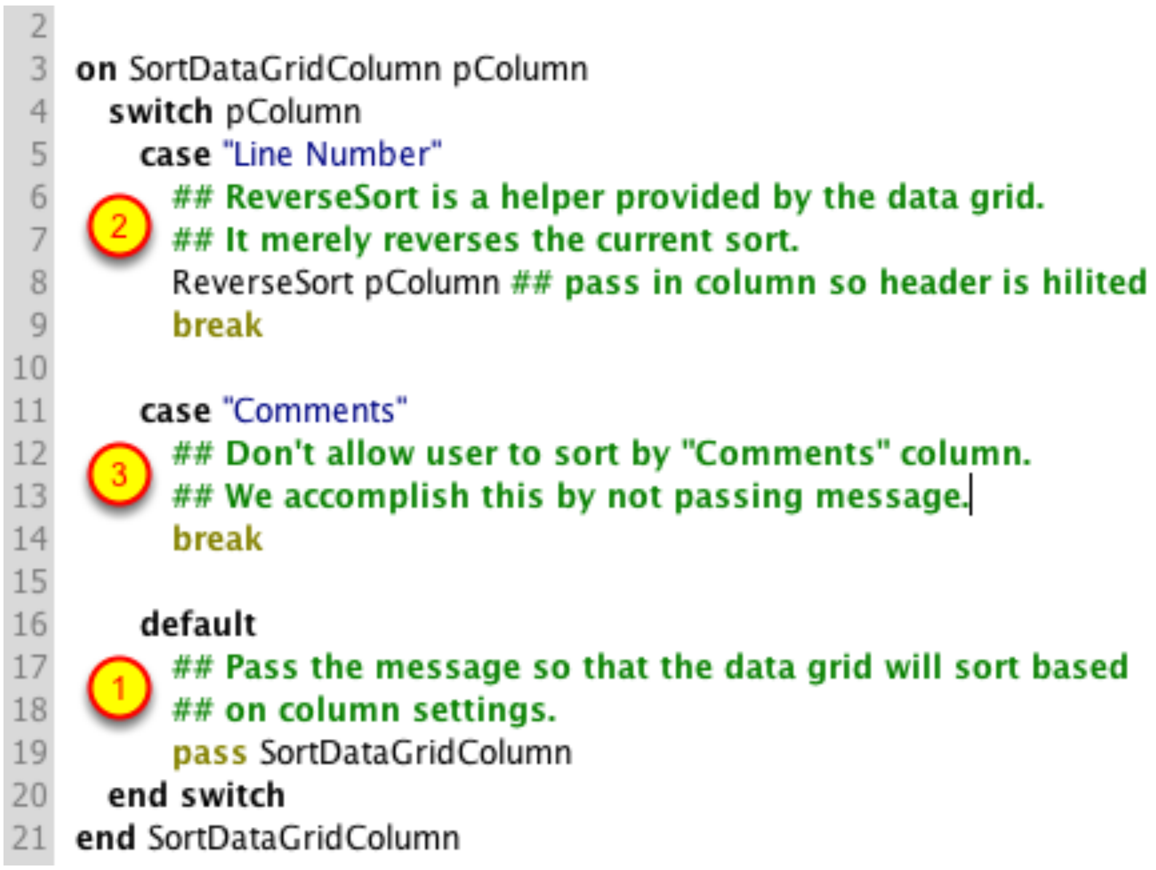
### A Data Grid Table



| Line Number | Name | Artist | Composer | Album | Grou |
|---|---|---|---|---|---|
| 1 | Crawlin' Kingsnal | John Lee Hooker | | The Definitive Co | |
| 2 | Wheel And Deal | John Lee Hooker | | The Definitive Co | |
| 3 | I'm Going Upstair | John Lee Hooker | | The Definitive Co | |
| 4 | The Road Is So Ro | John Lee Hooker | | The Definitive Co | |
| 5 | Boom Boom | John Lee Hooker | | The Definitive Co | |
| 6 | I'm Mad Again | John Lee Hooker | | The Definitive Co | |
| 7 | Boogie Chillun | John Lee Hooker | | The Definitive Co | |
| 8 | Don't Look Back | John Lee Hooker | | The Definitive Co | |
| 9 | Mama You Got A | John Lee Hooker | | The Definitive Co | |
| 10 | Solid Sender | John Lee Hooker | | The Definitive Co | |
| 11 | Want Ad Blues | John Lee Hooker | | The Definitive Co | |
| 12 | Hard Headed Wor | John Lee Hooker | | The Definitive Co | |
| 13 | This Is Hip | John Lee Hooker | | The Definitive Co | |
| 14 | It Serves Me Righ | John Lee Hooker | | The Definitive Co | |
| 15 | She's Long, She's | John Lee Hooker | | The Definitive Co | |
| 16 | Someday | John Legend | J. Stephens | August Rush (Mu | |
| 17 | No Such Thing (L | John Mayer | Clay Cook & John | Any Given Thurs | |
| 18 | Message in a Bott | John Mayer | Sting | Any Given Thurs | |

I am going to customize when happens when this table is sorted by the **Line Number** column.

## The Data Grid Group Script



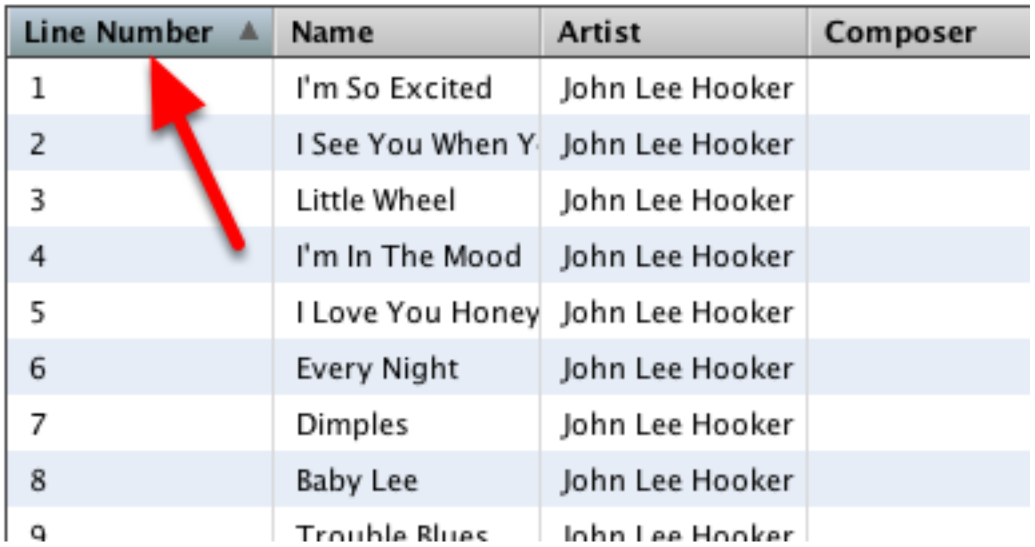
```
on SortDataGridColumn pColumn
  switch pColumn
    case "Line Number"
      ## ReverseSort is a helper provided by the data grid.
      ## It merely reverses the current sort.
      ReverseSort pColumn ## pass in column so header is hilited
      break

    case "Comments"
      ## Don't allow user to sort by "Comments" column.
      ## We accomplish this by not passing message.
      break

    default
      ## Pass the message so that the data grid will sort based
      ## on column settings.
      pass SortDataGridColumn
  end switch
end SortDataGridColumn
```

In the script for the data grid I added the above handler. **SortDataGridColumn** is called whenever the dgProps["sort by column"] property is set. This includes when the user clicks on a column header to sort.

The way SortDataGridColumn works is simple. If you pass the message then the data grid will use the default column sorting routines (1). If you don't pass the message then the data grid will not perform any sorting (2) and (3).

I want **Line Number** to just reverse the current sort so I call ReverseSort which is a built-in data grid helper that reverses whatever the current sort order is(2).

## The Result



| Line Number ▲ | Name | Artist | Composer |
|---|---|---|---|
| 1 | I'm So Excited | John Lee Hooker | |
| 2 | I See You When Y | John Lee Hooker | |
| 3 | Little Wheel | John Lee Hooker | |
| 4 | I'm In The Mood | John Lee Hooker | |
| 5 | I Love You Honey | John Lee Hooker | |
| 6 | Every Night | John Lee Hooker | |
| 7 | Dimples | John Lee Hooker | |
| 8 | Baby Lee | John Lee Hooker | |
| 9 | Trouble Blues | John Lee Hooker | |

Clicking on the **Line Number** column now reverses the sort.

## Custom Sort Example

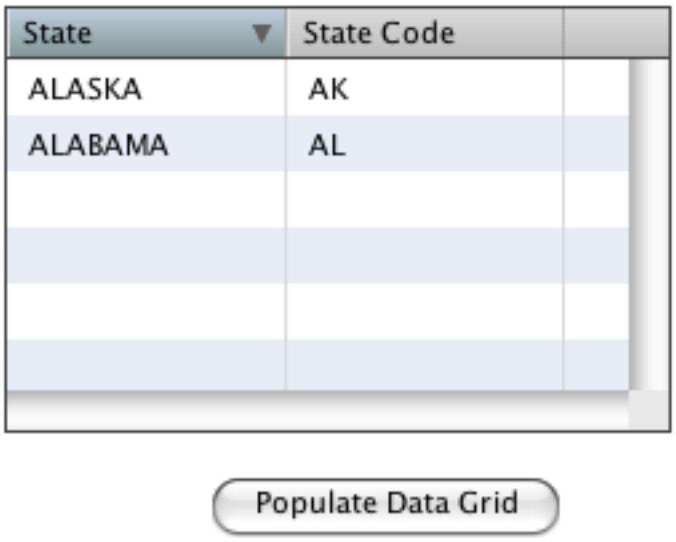
```
## Place in Data Grid script.

## This example shows how to perform a custom sort of data. This is similar
## to the internal routine that the data grid uses.

on SortDataGridColumn pColumn
   ## Begin by building a list of lines with two items (tab delimited)
   ## The first item is the index of each record stored in the data grid.
   ## The second item is the key you are going to sort on.
   put the dgData of me into theDataA
   repeat for each key theIndex in theDataA
      put theIndex & tab & theDataA[theIndex][pColumn] & cr after theData
   end repeat
   delete the last char of theData
   set the itemdelimiter to tab

   ## perform sort. Lookup the current sort direction for this column
   ## and use that as sort direction
   put the dgColumnSortDirection[pColumn] of me into theSortDirection
   if theSortDirection is "ascending" then
      sort lines of theData ascending by item 2 to -1 of each    #
   else
      sort lines of theData descending by item 2 to -1 of each
```

```
    end if

    ## Rebuild the order of indexes in the data grid
    put empty into theIndexSequencing
    repeat for each line theLine in theData
        put item 1 of theLine & comma after theIndexSequencing
    end repeat
    delete the last char of theIndexSequencing

    ## Set the dgIndexes property to new order
    set the dgIndexes of me to theIndexSequencing

    ## Tell data grid to hilite column
    HiliteAndStoreSortByColumn pColumn
end SortDataGridColumn
```

## How Do I Disable Column Sorting?

This lesson will show you how to disable column sorting in a Data Grid table.

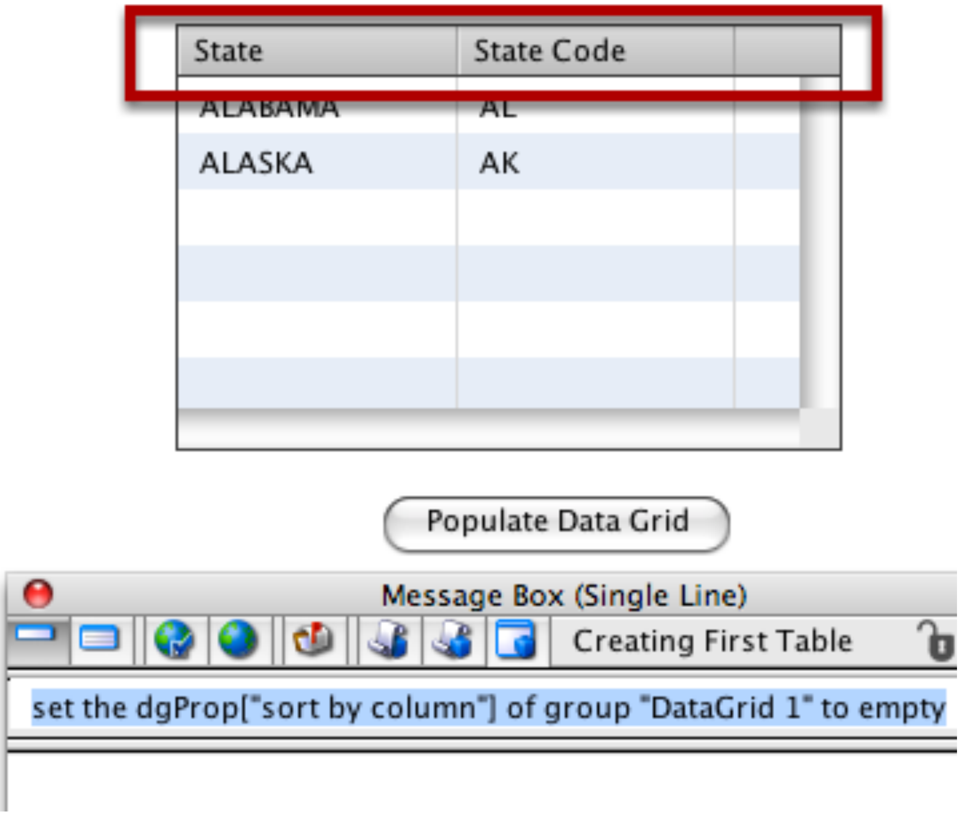
### Sort By No Column



| State | State Code |
|---|---|
| ALASKA | AK |
| ALABAMA | AL |

To begin, make sure you set the **sort by column** property to empty. This will remove any column sorting that might be active.

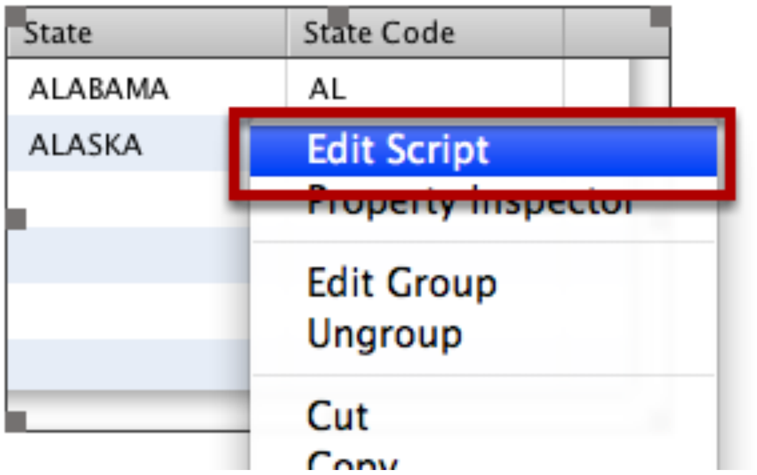
```
set the dgProp["sort by column"] of group "DataGrid 1" to empty
```



```
set the dgProp["sort by column"] of group "DataGrid 1" to empty
```

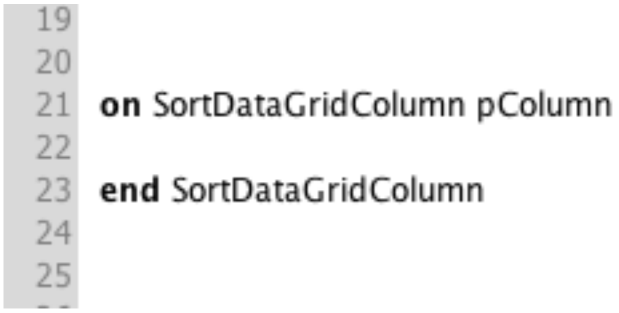
Your table headers should now appear the same, with no highlights or sorting arrow.

## Edit Data Grid Script



Edit the Data Grid group script by right clicking on the Data Grid and selecting **Edit Script**.

## Add SortDataGridColumn Handler



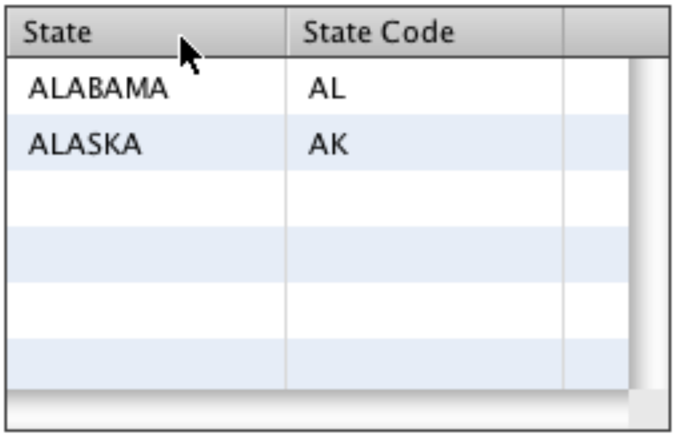
```
on SortDataGridColumn pColumn

end SortDataGridColumn
```

When the user clicks on a table column header the **SortDataGridColumn** message is sent to the Data Grid. By intercepting the message and not passing it you will effectively disable sorting.  For more information about how SortDataGridColumn works please see this lesson.

Add the following code to your Data Grid group script and compile.

**==========**
***Copy & Paste***
**==========**
**on** SortDataGridColumn pColumn

**end** SortDataGridColumn



| State | State Code |
|---|---|
| ALABAMA | AL |
| ALASKA | AK |

Now when you click on the table header the data will not be sorted.

**Note:** Once you add the SortDataGridColumn handler to a Data Grid setting the dgProp["sort by column"] property will no longer do anything. If you need to set the "sort by column" property later on you would need to remove the SortDataGridColumn handler from the script, or at least pass the message.

## How Do I Perform An Action After the User Sorts a Data Grid?

This lesson will show you how to modify a data grid script so that you can perform an action whenever a data grid is sorted.

### Add SortDataGridColumn To Data Grid Group Script

You can customize what happens when a data grid is sort by adding a **SortDataGridColumn** handler to your data grid group script. For more information about how SortDataGridColumn works please see this lesson.

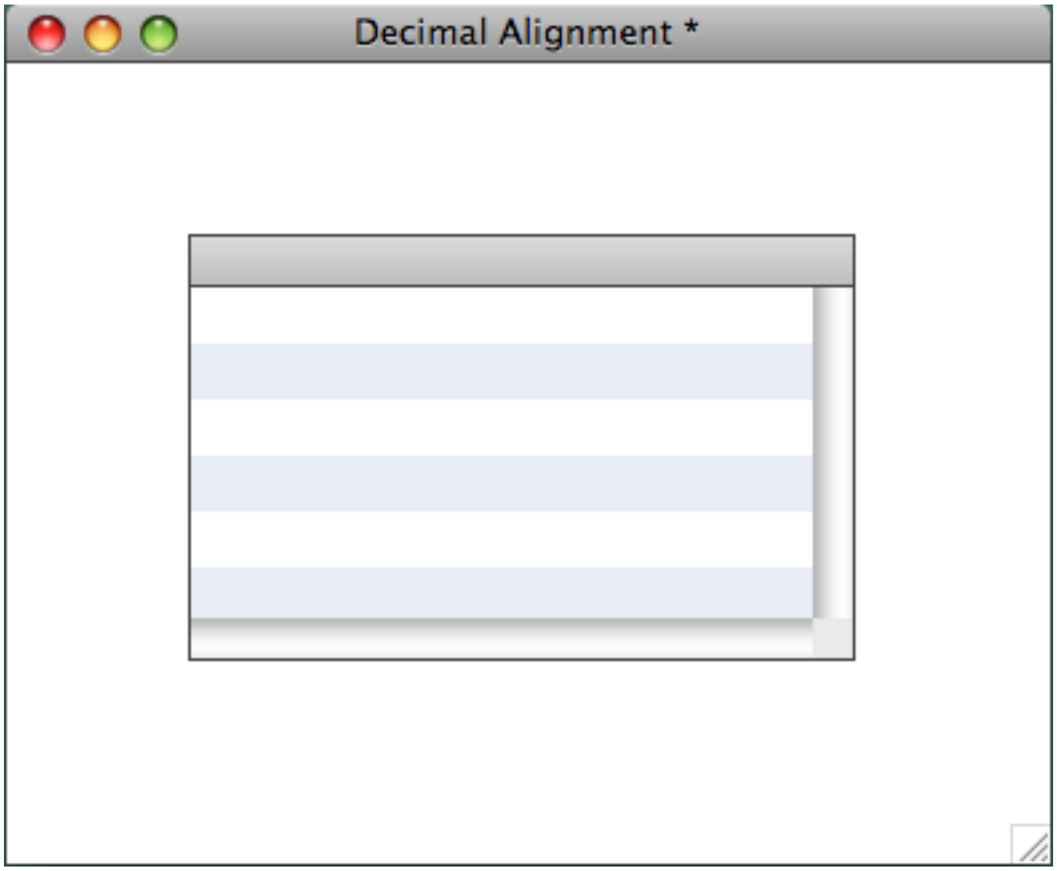
In order to perform an action AFTER the sort has been applied you simply need to call the handler the Data Grid behavior uses to sort but NOT pass the SortDataGridColumn handler. Doing so sorts the data just like the Data Grid would and then perform any additional actions you would like.

```
## Place in Data Grid script.
on SortDataGridColumn pColumn
   ## Call Data Grid sort routine by hand
   SortByColumn pColumn

   ## Now data has been sorted, do whatever you need to do.
   ...

   ## DON'T PASS!
end SortDataGridColumn
```

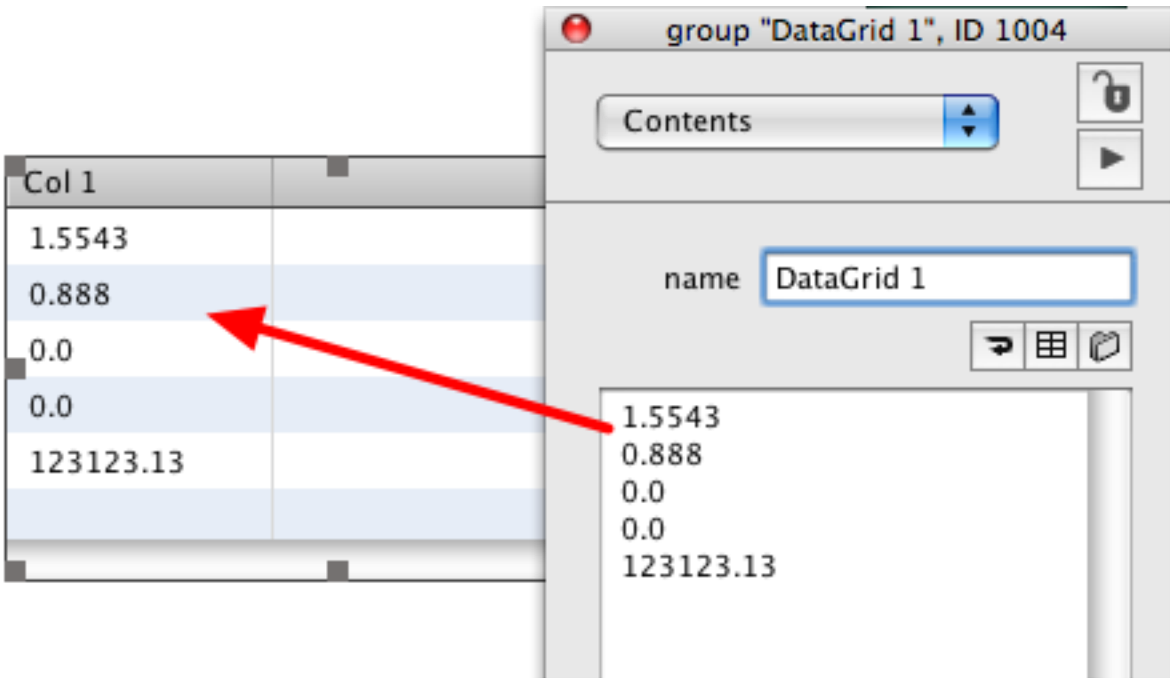
## How Do I Align Decimals in a Column?

This lesson will show you how to create a custom column template that provides decimal alignment. You can download the stack used to create this lesson.

### Create a Data Grid Table



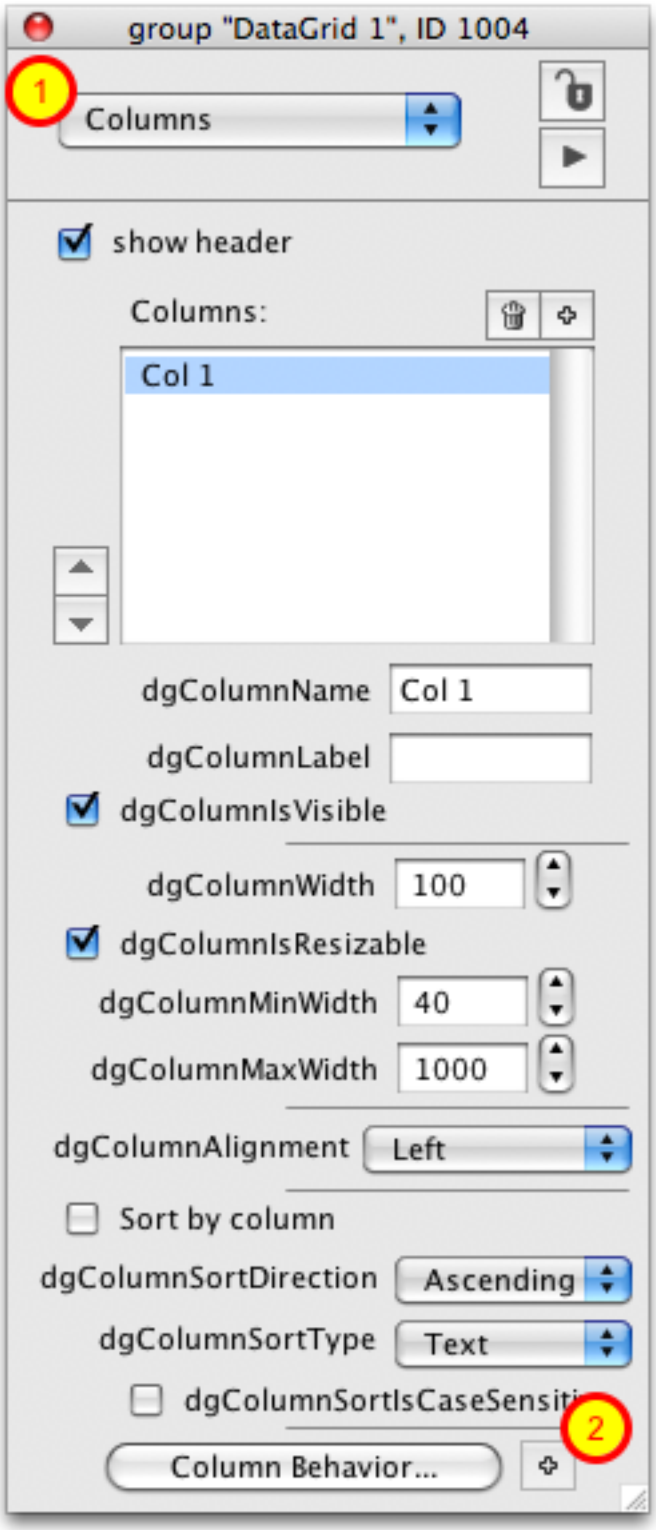
## Add Some Data



```
1.5543
0.888
0.0
0.0
123123.13
```

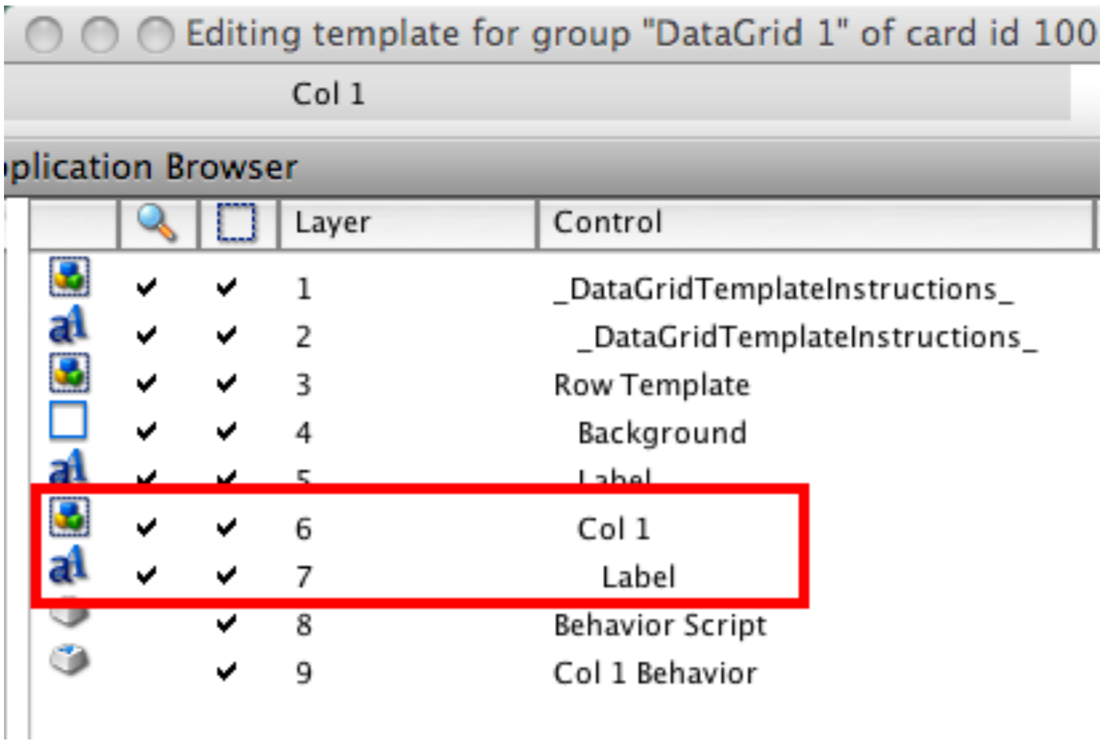
I've added a couple of rows of decimal numbers to experiment with.

## Create Column Template



Select the **Columns** pane (1) in the property inspector and click the **Add custom column behavior** button (2). This will create a custom column template and behavior script.

## Edit Column Template


Editing template for group "DataGrid 1" of card id 100

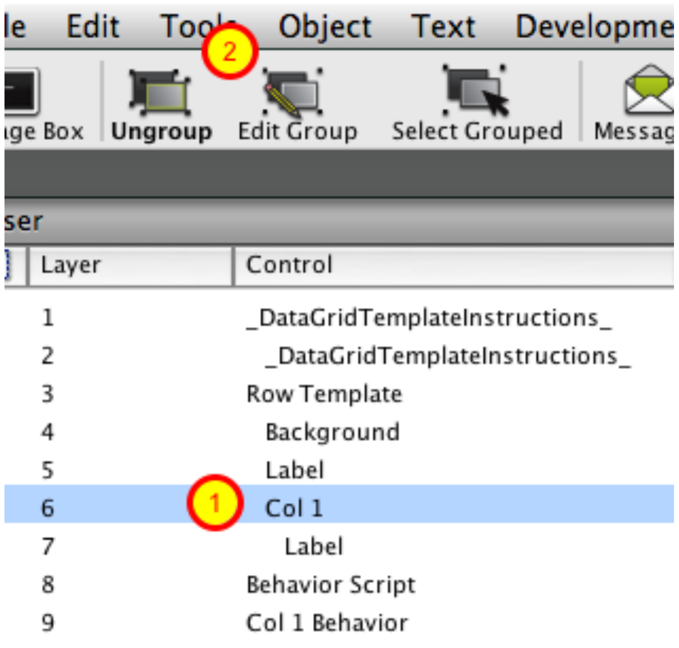
Col 1

plication Browser

|  | 🔍 | ⬚ | Layer | Control |
|---|---|---|---|---|
|  | ✓ | ✓ | 1 | _DataGridTemplateInstructions_ |
|  | ✓ | ✓ | 2 | _DataGridTemplateInstructions_ |
|  | ✓ | ✓ | 3 | Row Template |
|  | ✓ | ✓ | 4 | Background |
|  | ✓ | ✓ | 5 | Label |
|  | ✓ | ✓ | 6 | Col 1 |
|  | ✓ | ✓ | 7 | Label |
|  |  | ✓ | 8 | Behavior Script |
|  |  | ✓ | 9 | Col 1 Behavior |

When you create a custom template for a column a new group, named after the column, is added to the data grid **row template.** Here you see that I have selected the card the row template is on (the card is part of the **Data Grid XXX** substack of your stack) and I can see the group for my column.

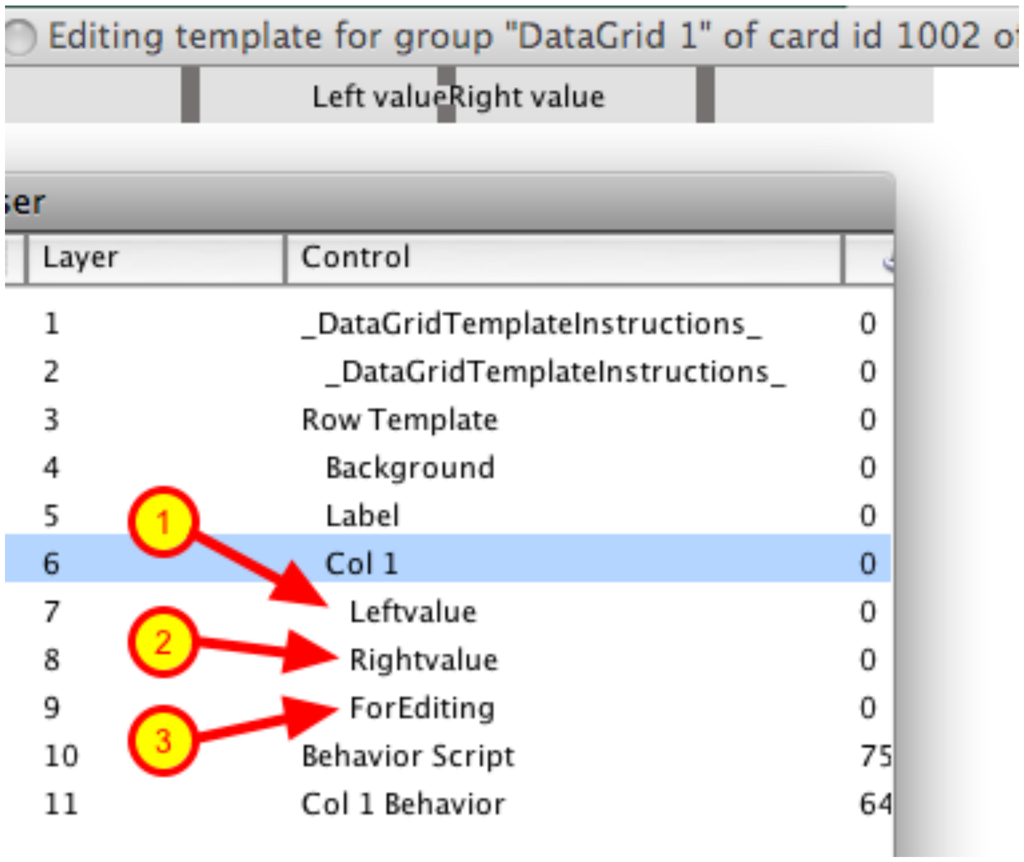
## Edit Column Template Group



| Layer | Control |
|---|---|
| 1 | _DataGridTemplateInstructions_ |
| 2 | _DataGridTemplateInstructions_ |
| 3 | Row Template |
| 4 | Background |
| 5 | Label |
| 6 | Col 1 |
| 7 | Label |
| 8 | Behavior Script |
| 9 | Col 1 Behavior |

Select the column group in the Application Browser (1) and click the **Edit Group** button (2) to edit the contents of the column group.

## Create 3 Fields


Editing template for group "DataGrid 1" of card id 1002 o

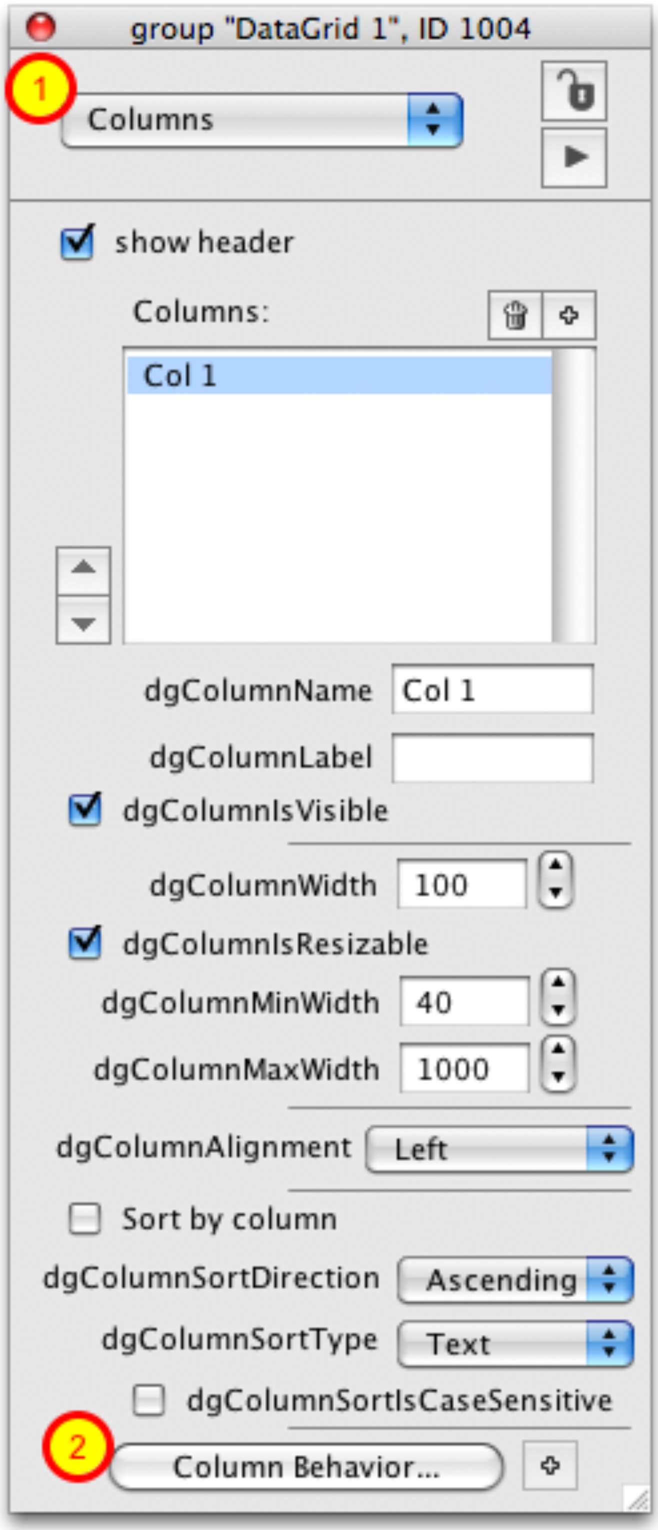
Left valueRight value

| Layer | Control | |
|---|---|---|
| 1 | _DataGridTemplateInstructions_ | 0 |
| 2 | _DataGridTemplateInstructions_ | 0 |
| 3 | Row Template | 0 |
| 4 | Background | 0 |
| 5 | Label | 0 |
| 6 | Col 1 | 0 |
| 7 | Leftvalue | 0 |
| 8 | Rightvalue | 0 |
| 9 | ForEditing | 0 |
| 10 | Behavior Script | 75 |
| 11 | Col 1 Behavior | 64 |

A column that aligns decimals in the center of the column, and is editable if the user double-clicks on the cell, is achieved by creating three fields. Since the column template group already had a field named "Label" I **renamed** it to "Leftvalue" (1). Set the **textalign** of this field to "right". **Duplicate** the "Leftvalue" field and name it "Rightvalue" (2). Set the **textalign** to right. **Duplicate** this field and **rename** the duplicate to "ForEditing" (3). The text of this field should be empty. Make sure that the **ForEditing** field has a higher layer than **Leftvalue** or **Rightvalue.**

Don't worry too much about positioning as the fields will be resized and repositioned when drawn in the data grid.

## Edit Column Behavior


group "DataGrid 1", ID 1004

Columns

show header [✓]

Columns:

Col 1

dgColumnName: Col 1

dgColumnLabel:

dgColumnIsVisible [✓]

dgColumnWidth: 100

dgColumnIsResizable [✓]

dgColumnMinWidth: 40

dgColumnMaxWidth: 1000

dgColumnAlignment: Left

Sort by column [ ]

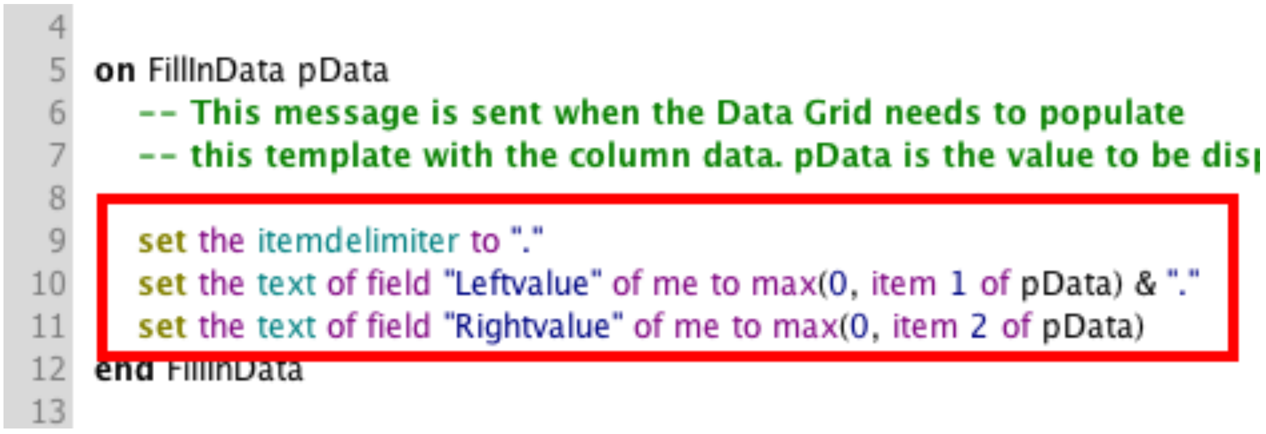
dgColumnSortDirection: Ascending

dgColumnSortType: Text

dgColumnSortIsCaseSensitive [ ]

Now that we have created the objects we need to display the column data we need to tell the column what to do when it is drawn. Return to the Property Inspector and from the **Columns** pane (1) click the **Column Behavior** button (2). This will open the behavior script for the column.

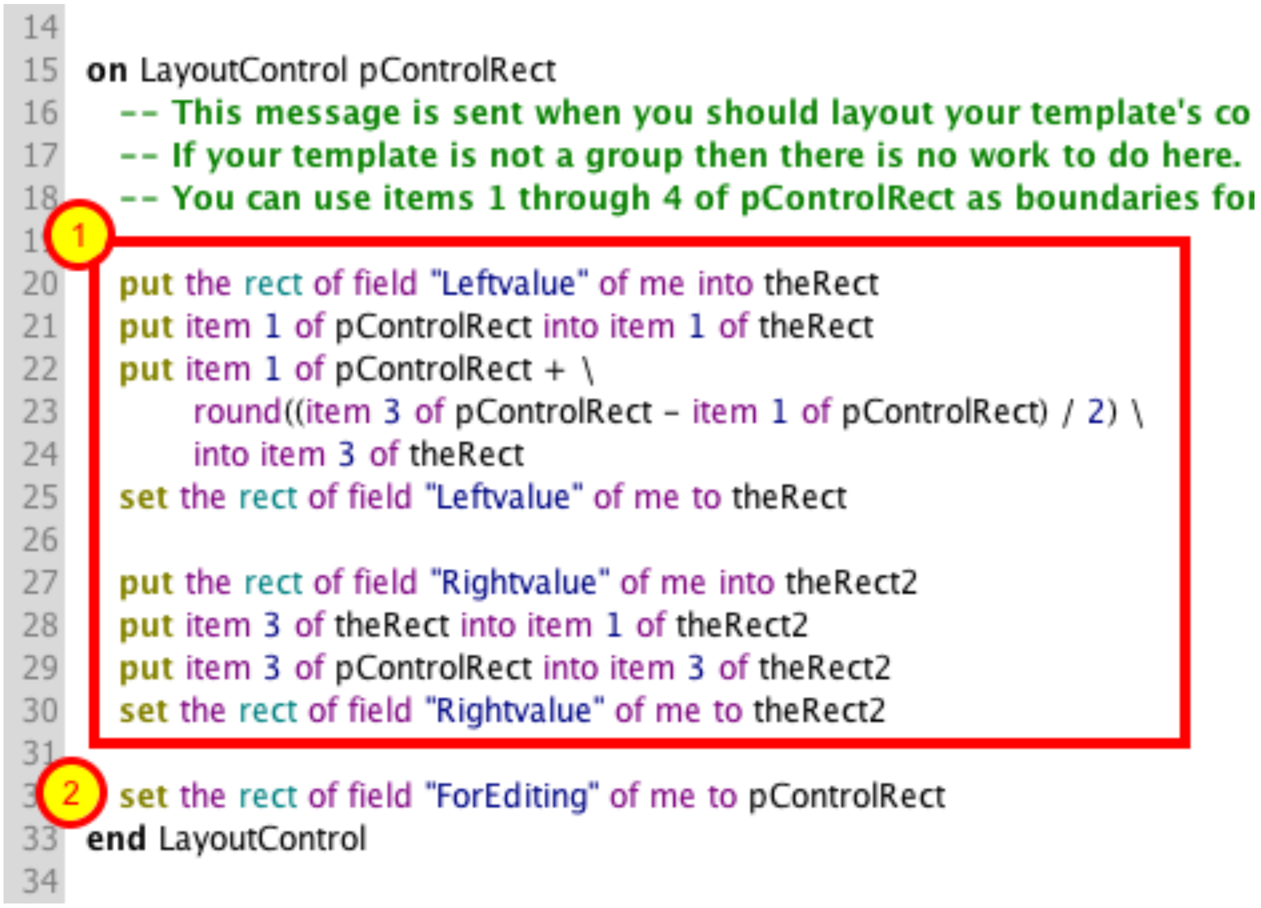
## Update FillInData



```
on FillInData pData
   -- This message is sent when the Data Grid needs to populate
   -- this template with the column data. pData is the value to be dis

   set the itemdelimiter to "."
   set the text of field "Leftvalue" of me to max(0, item 1 of pData) & "."
   set the text of field "Rightvalue" of me to max(0, item 2 of pData)
end FillInData
```

The decimal number is going to be displayed by assigning the left side of the decimal to the **Leftvalue** field and the right side of the decimal to the **Rightvalue** field.

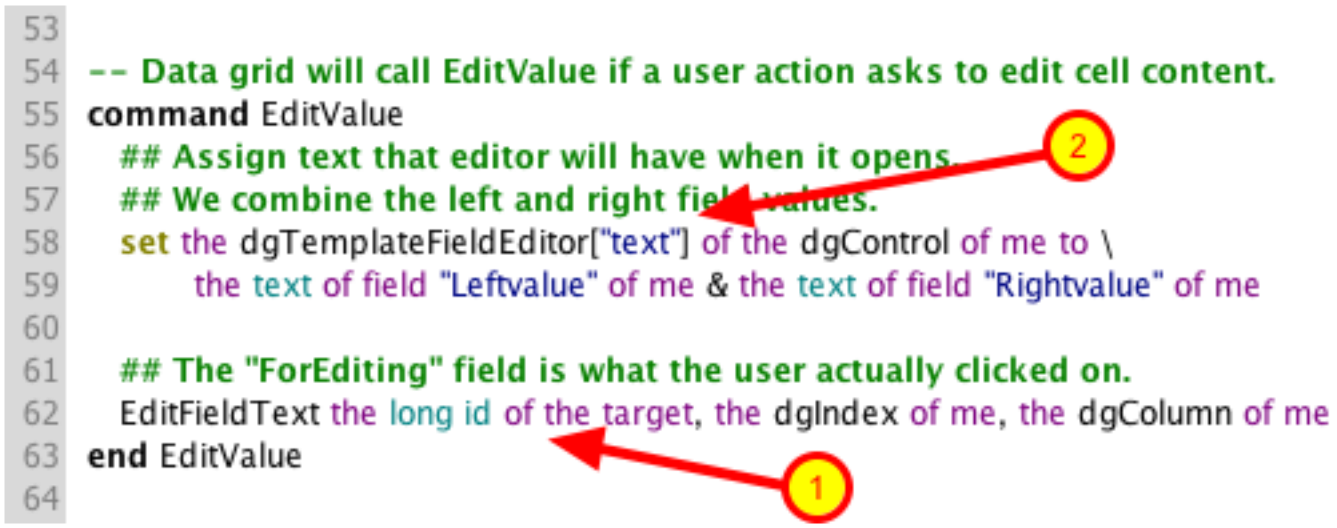
## Update LayoutControl



```
on LayoutControl pControlRect
   -- This message is sent when you should layout your template's co
   -- If your template is not a group then there is no work to do here.
   -- You can use items 1 through 4 of pControlRect as boundaries fo

   put the rect of field "Leftvalue" of me into theRect
   put item 1 of pControlRect into item 1 of theRect
   put item 1 of pControlRect + \
         round((item 3 of pControlRect - item 1 of pControlRect) / 2) \
         into item 3 of theRect
   set the rect of field "Leftvalue" of me to theRect

   put the rect of field "Rightvalue" of me into theRect2
   put item 3 of theRect into item 1 of theRect2
   put item 3 of pControlRect into item 3 of theRect2
   set the rect of field "Rightvalue" of me to theRect2

   set the rect of field "ForEditing" of me to pControlRect
end LayoutControl
```

Since we want to align the decimal in the center of the cell we resize the left and right fields so that each takes up half of the available width (1). Finally we resize the **ForEditing** field to take up the entire cell (2). This is the field that the user can double-click on to edit the cell contents.

## Update EditValue

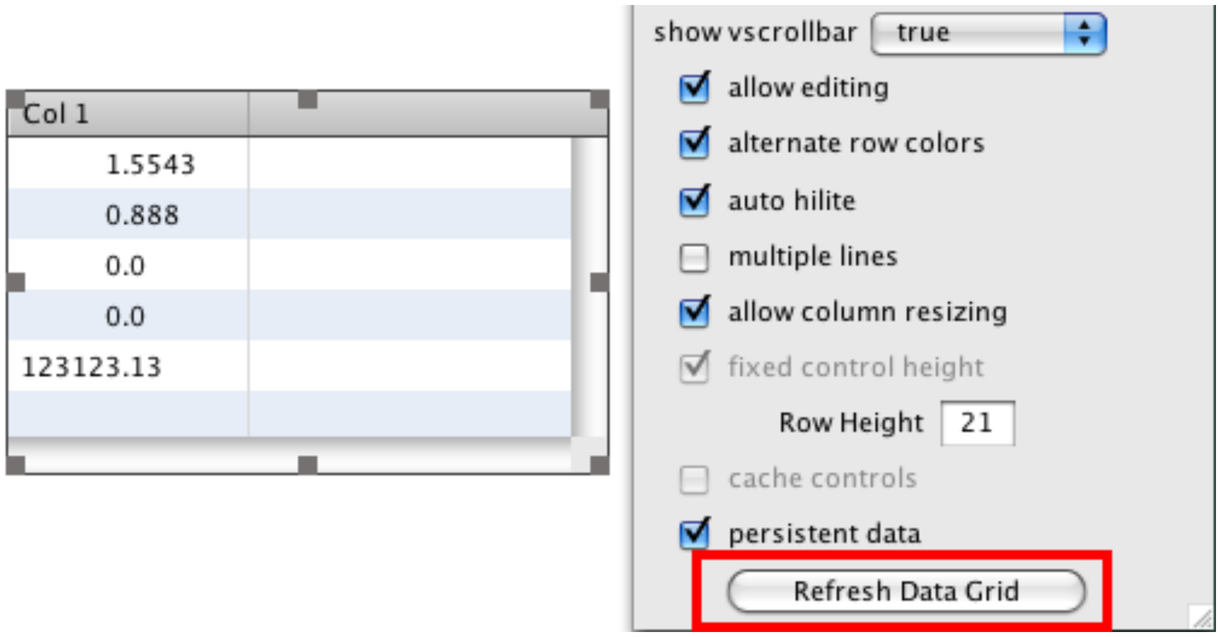


```
-- Data grid will call EditValue if a user action asks to edit cell content.
command EditValue
   ## Assign text that editor will have when it opens.
   ## We combine the left and right field values.
   set the dgTemplateFieldEditor["text"] of the dgControl of me to \
         the text of field "Leftvalue" of me & the text of field "Rightvalue" of me

   ## The "ForEditing" field is what the user actually clicked on.
   EditFieldText the long id of the target, the dgIndex of me, the dgColumn of me
end EditValue
```

EditValue is the handler that is called when the data grid wants to edit a cell value. Usually this is because the user double-clicked on the cell or tabbed into the cell. Remember that if the user double-clicks on this cell they actually double-click on the **ForEditing** field. The problem is that the **ForEditing** field has no text in it and by default **EditFieldText** displays the htmltext of the field you are editing (1).

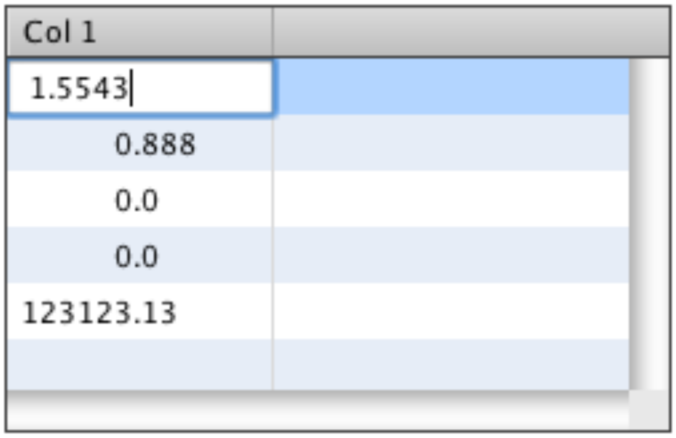
We can override this default by setting the dgTemplateFieldEditor[ text | htmltext | rtftext | unicodetext | utf8text ] of the data grid. Here we set the dgTemplateFieldEditor["text"] of the data grid (2).

## Refresh Data Grid



Now that we have finished making the necessary changes we need to refresh the data grid so that it redraws with the new template controls we've added. From the **Basic Properties** pane of the inspector palette click the **Refresh Data Grid** button. Alternatively you can send "ResetList" to the data grid.

## Test



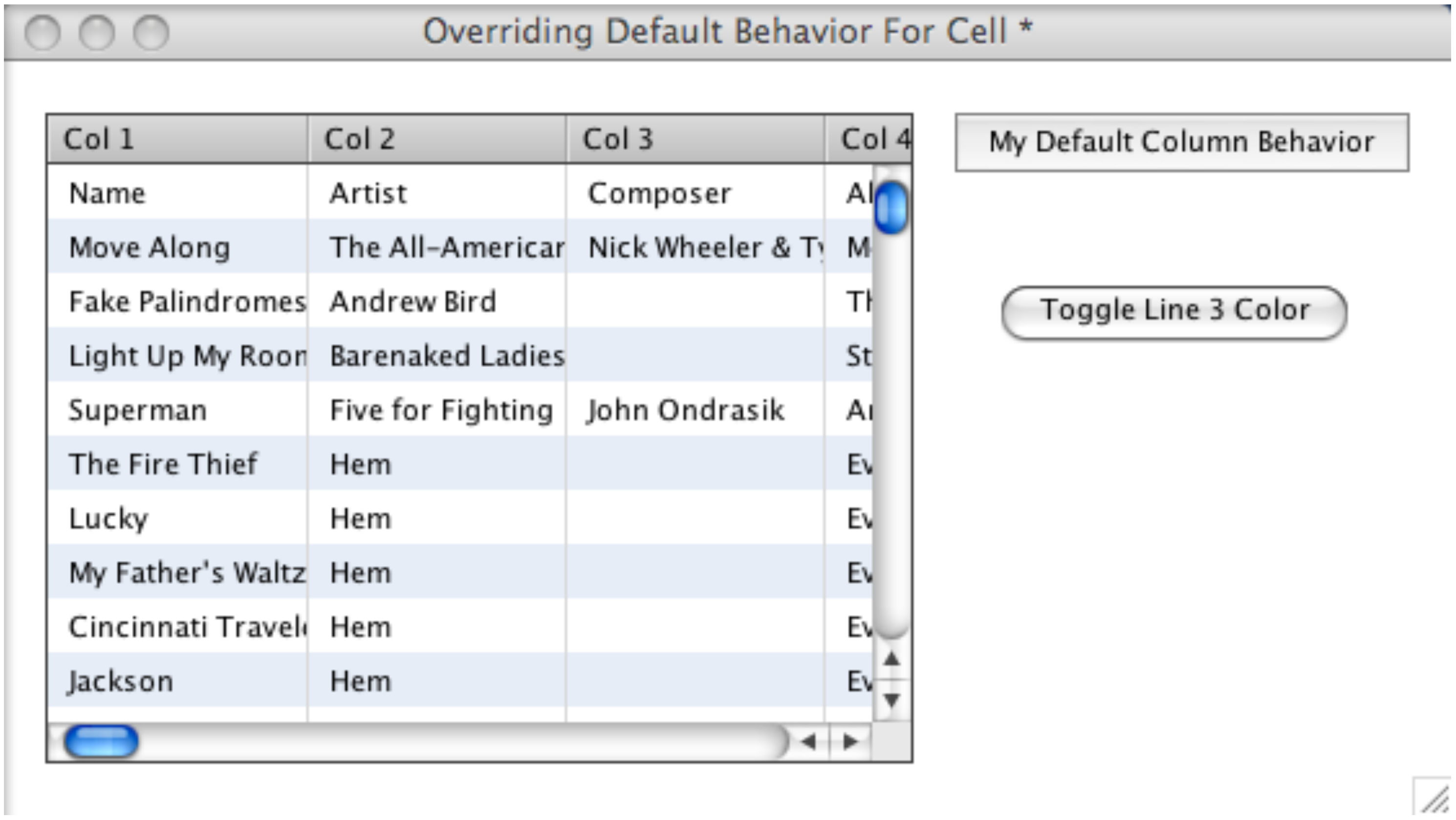
| Col 1 | |
|---|---|
| 1.5543 | |
| 0.888 | |
| 0.0 | |
| 0.0 | |
| 123123.13 | |
| | |

We can see that the decimals all appear in the center of the cell. Double-clicking brings up an editing field like you see here. If you resize the column you will notice that the decimal remains in the center of the cell.

## How Can I Colorize Individual Lines in a Table?

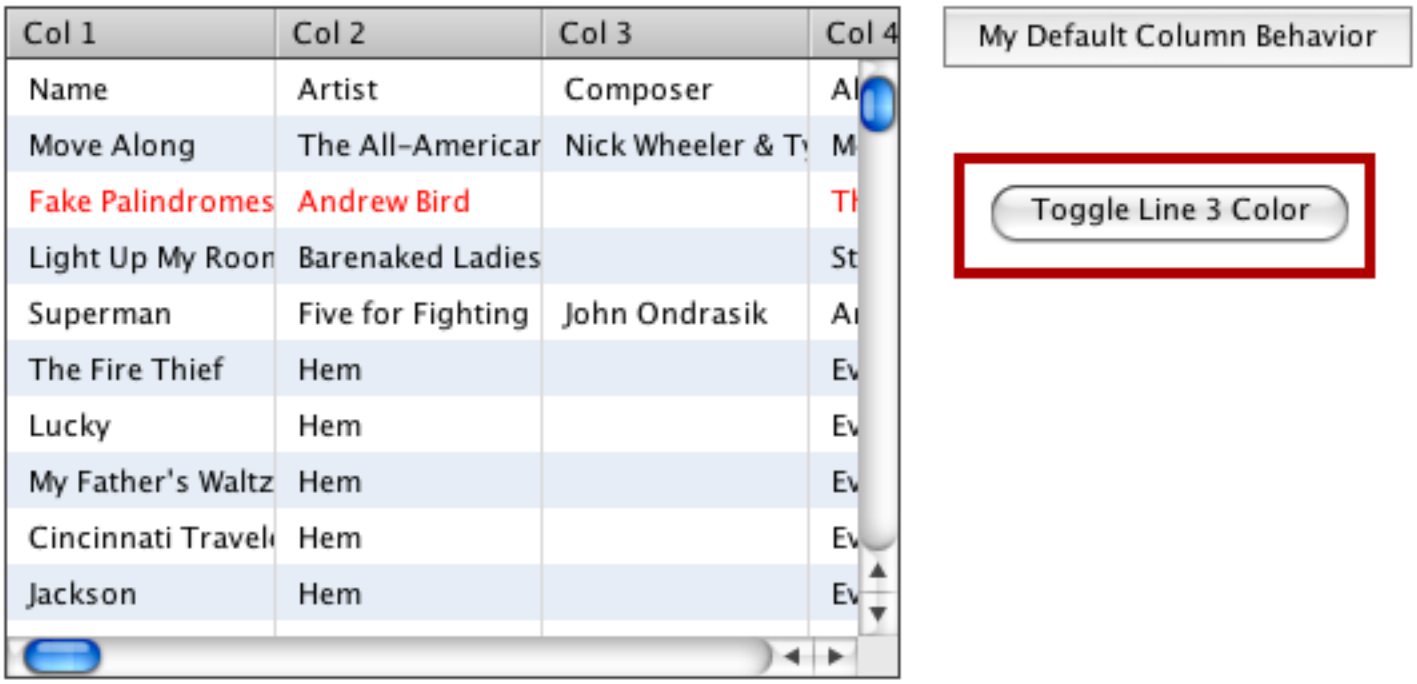
This lesson will show you how to change the color of individual lines in a Data Grid table. Before you being this lesson you will need to read the lesson How Do I Override the Default Behavior for Rendering Data to a Cell?

### Coloring A Single Line


Overriding Default Behavior For Cell *
My Default Column Behavior
Toggle Line 3 Color

| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| Name | Artist | Composer | Al |
| Move Along | The All-Americar | Nick Wheeler & Ty | M |
| Fake Palindromes | Andrew Bird | | Th |
| Light Up My Roon | Barenaked Ladies | | St |
| Superman | Five for Fighting | John Ondrasik | Ar |
| The Fire Thief | Hem | | Ev |
| Lucky | Hem | | Ev |
| My Father's Waltz | Hem | | Ev |
| Cincinnati Travel | Hem | | Ev |
| Jackson | Hem | | Ev |

In this example I am going to modify the basic example created in the lesson mentioned above so that it colors a row red if the row's *line has error* property is true.



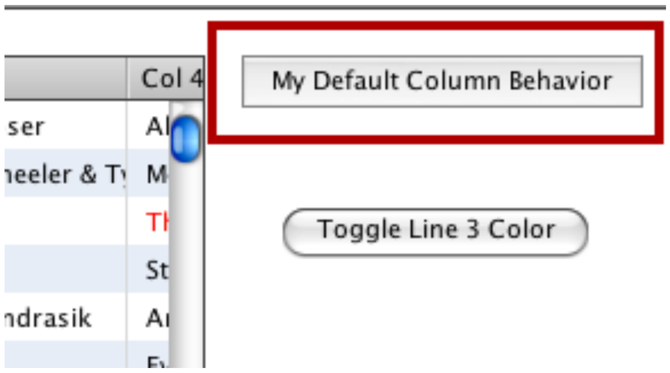
| Col 1 | Col 2 | Col 3 | Col 4 |
|---|---|---|---|
| Name | Artist | Composer | Al |
| Move Along | The All-Americar | Nick Wheeler & Ty | M |
| Fake Palindromes | Andrew Bird | | Th |
| Light Up My Roon | Barenaked Ladies | | St |
| Superman | Five for Fighting | John Ondrasik | Ar |
| The Fire Thief | Hem | | Ev |
| Lucky | Hem | | Ev |
| My Father's Waltz | Hem | | Ev |
| Cincinnati Travel | Hem | | Ev |
| Jackson | Hem | | Ev |

By clicking on the **Toggle Line 3 Color** button the color of line 3 turns red. The code in the button extracts the data for line 3 from the data grid, toggles the "line has error" property and reassigns the new data to line 3.

```
on mouseUp pMouseBtnNo
   put the dgDataOfLine[3] of group "DataGrid 1" into theDataA
   put not theDataA["line has error"] into theDataA["line has error"]
   set the dgDataOfLine[3] of group "DataGrid 1" to theDataA
end mouseUp
```

Now let's look at what the default custom column behavior looks like for this data grid.

## The Default Custom Column Behavior



Here is the relevant code in the default custom column behavior for this data grid. The key addition to the code is the **SetForeGroundColor** handler. This handler checks the value of the "line has error" column for this row (using GetDataOfIndex) and if it is true then changes the cell to red. Since each cell in this row has the same value for "line has error" the entire row appears red.

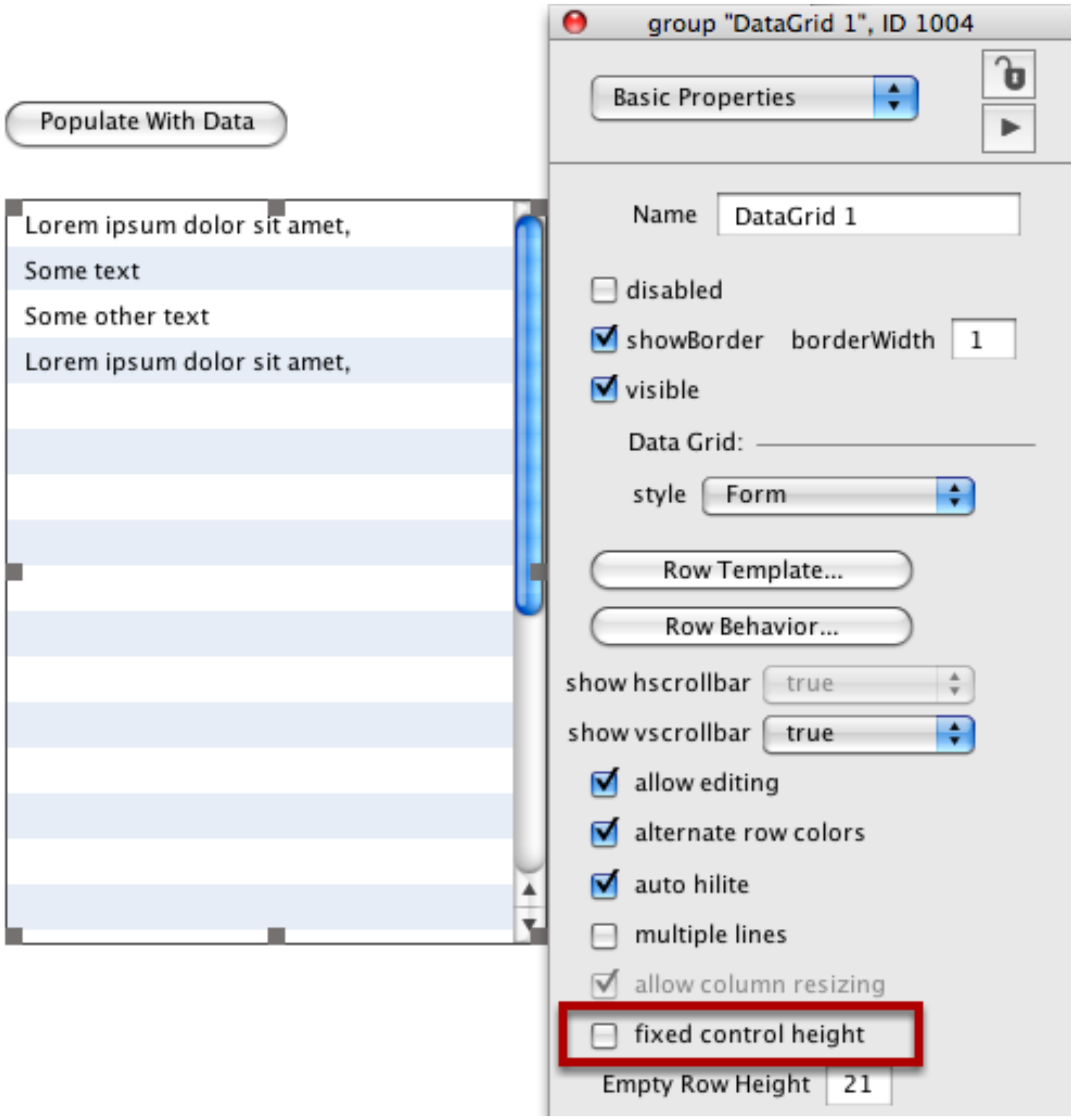
```
on FillInData pData
   -- This message is sent when the Data Grid needs to populate
   -- this template with the column data. pData is the value to be displayed.
   set the text of me to pData

   SetForeGroundColor
end FillInData


on LayoutControl pControlRect
   -- A default column is just a field.
end LayoutControl


setprop dgHilite pBoolean
   -- This custom property is set when the highlight of your column template has
   -- changed. You only add script here if you want to customize the highlight.
   if pBoolean then
      set the foregroundcolor of me to the dgProp["hilited text color"] of the dgControl of me
   else
      SetForeGroundColor
   end if
end dgHilite


private command SetForeGroundColor
   if GetDataOfIndex(the dgIndex of me, "line has error") then
      set the textcolor of me to red
   else
      set the textcolor of me to black
   end if
end SetForeGroundColor
```

# Working With Data Grid Forms

## How Do I Create a Form with Variable Line Heights?

This lesson will show you how to create a data grid form with variable height lines by modifying the default data grid **Row Template** and **Row Behavior**.

### Turn Off "fixed control height"



Name: DataGrid 1

- [ ] disabled
- [x] showBorder — borderWidth: 1
- [x] visible

Data Grid:

style: Form

show hscrollbar: true

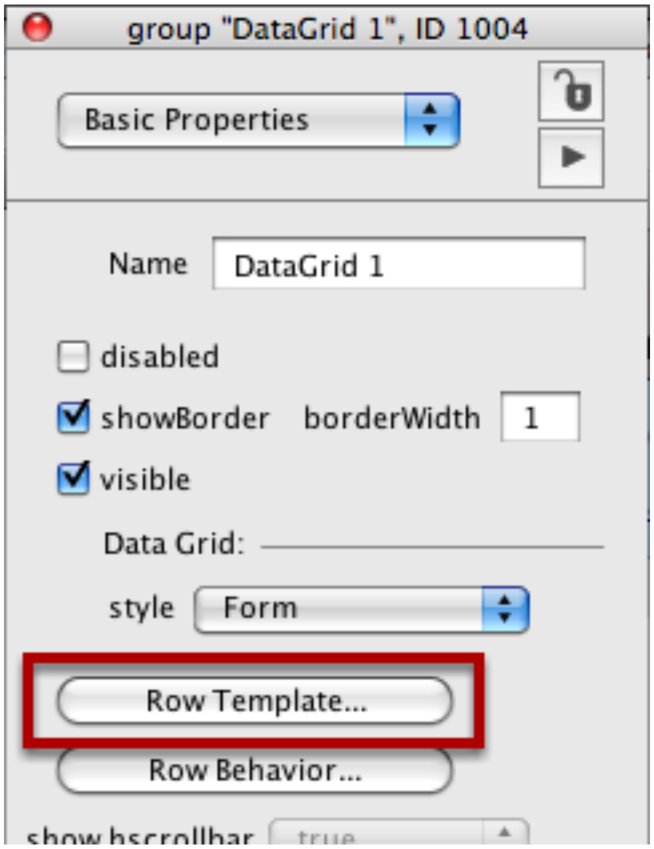
show vscrollbar: true

- [x] allow editing
- [x] alternate row colors
- [x] auto hilite
- [ ] multiple lines
- [x] allow column resizing
- [ ] fixed control height

Empty Row Height: 21

To begin, turn off **fixed control height** for your data grid form in the property inspector.

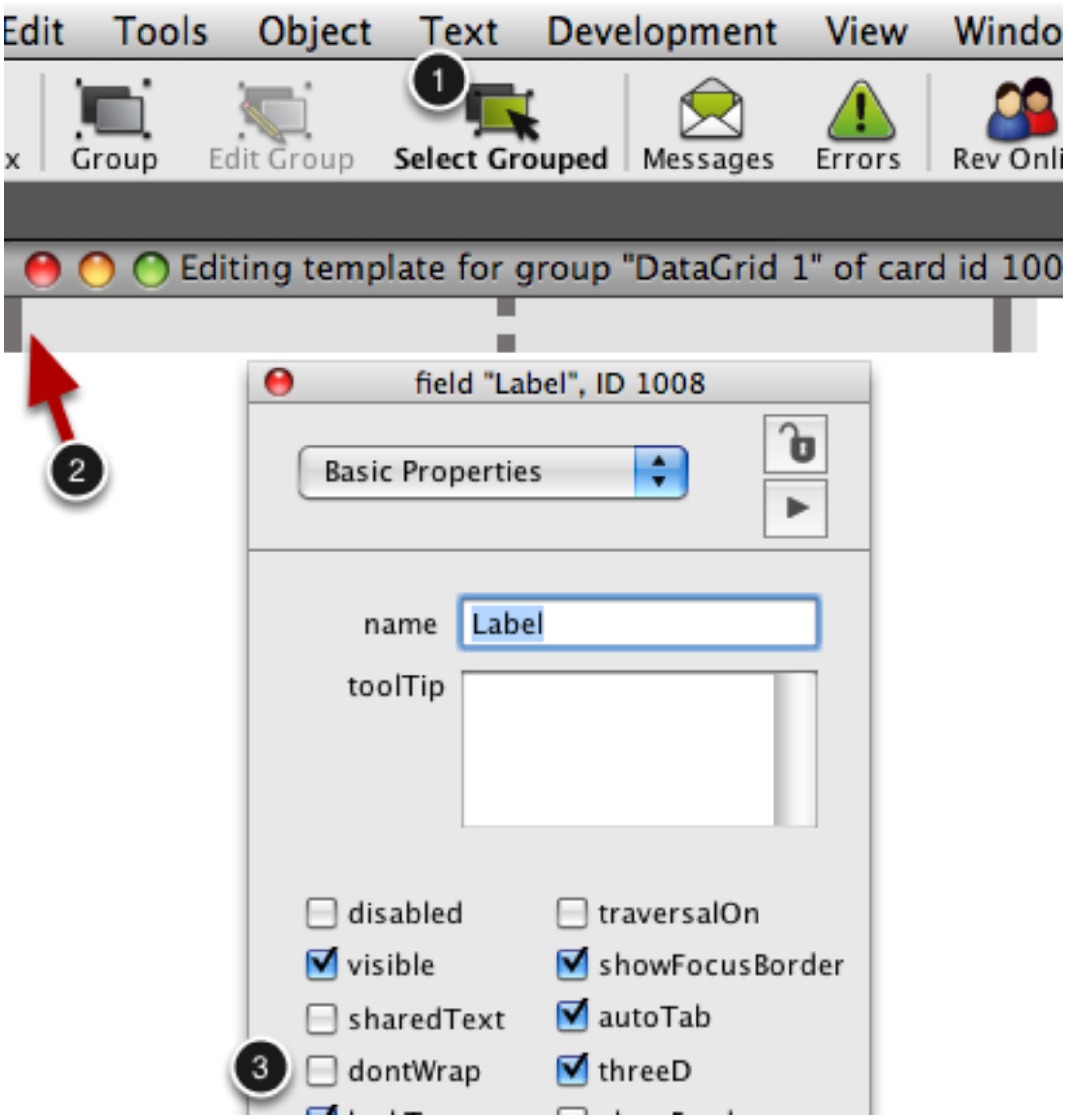
## Edit Row Template



Name: DataGrid 1
disabled: unchecked
showBorder: checked
borderWidth: 1
visible: checked
style: Form
show hscrollbar: true

Open the card that has the row template group by clicking the **Row Template** button.

## Turn Off dontWrap Property



Editing template for group "DataGrid 1" of card id 100

field "Label", ID 1008

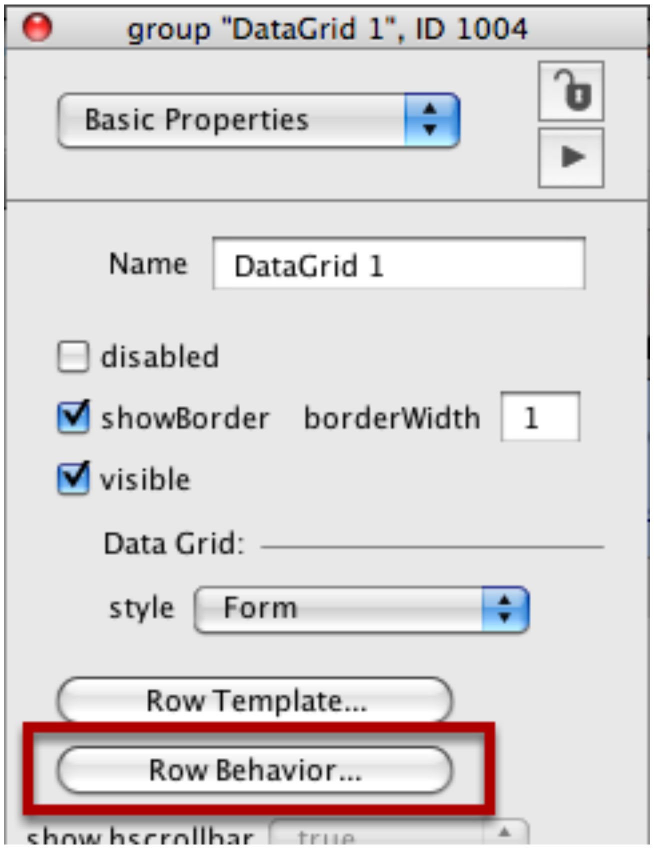
name: Label

toolTip:

- [ ] disabled
- [ ] traversalOn
- [✓] visible
- [✓] showFocusBorder
- [ ] sharedText
- [✓] autoTab
- [ ] dontWrap
- [✓] threeD

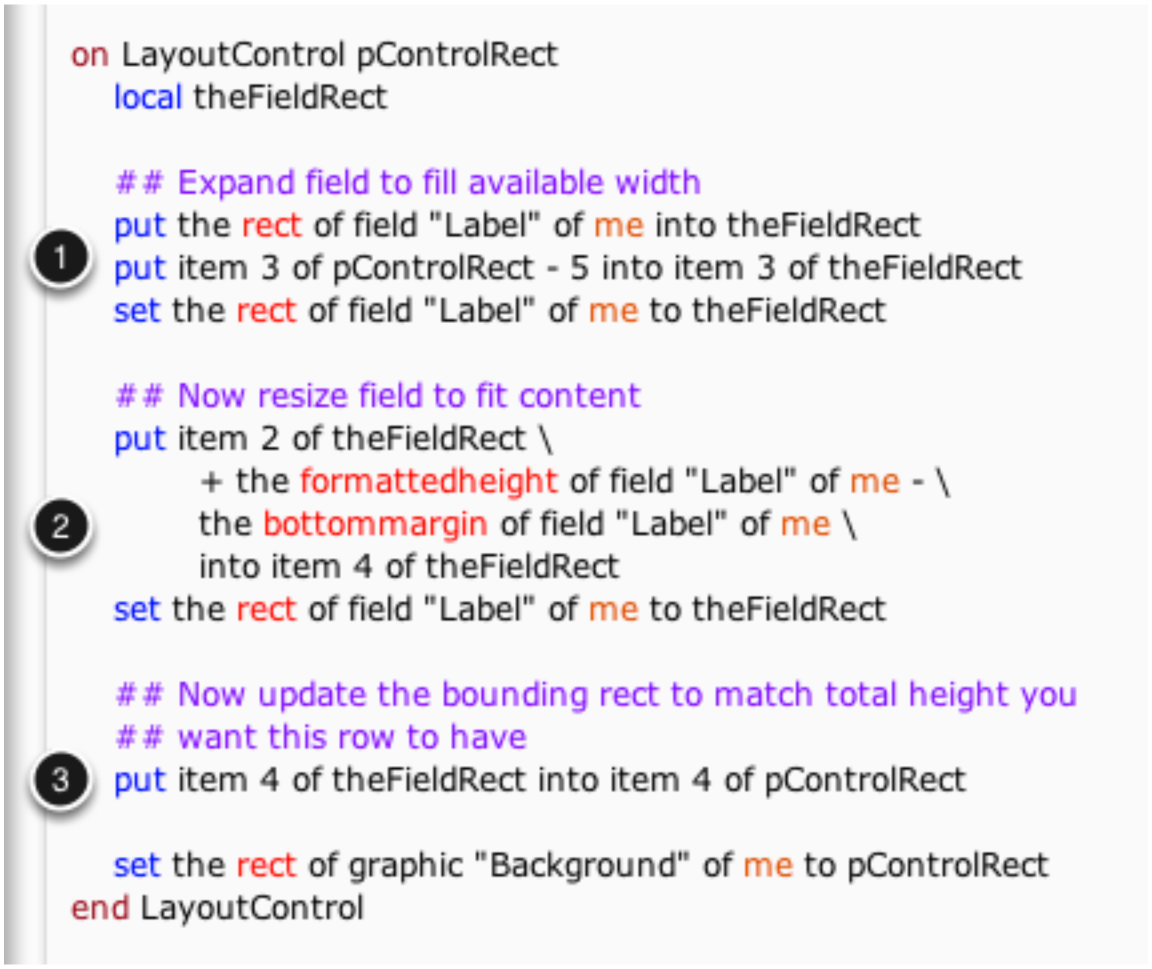
The default row template for a data grid has a single field that displays data. With **Select Grouped** turned on (1) select the field (2). Using the Property Inspector turn off **dontWrap** (3). You can now close the stack.

## Edit Row Behavior



Return to the Property Inspector for your data grid. Edit the **Row Behavior** by clicking on the Row Behavior button.

## Script Field to Resize to Fit Height

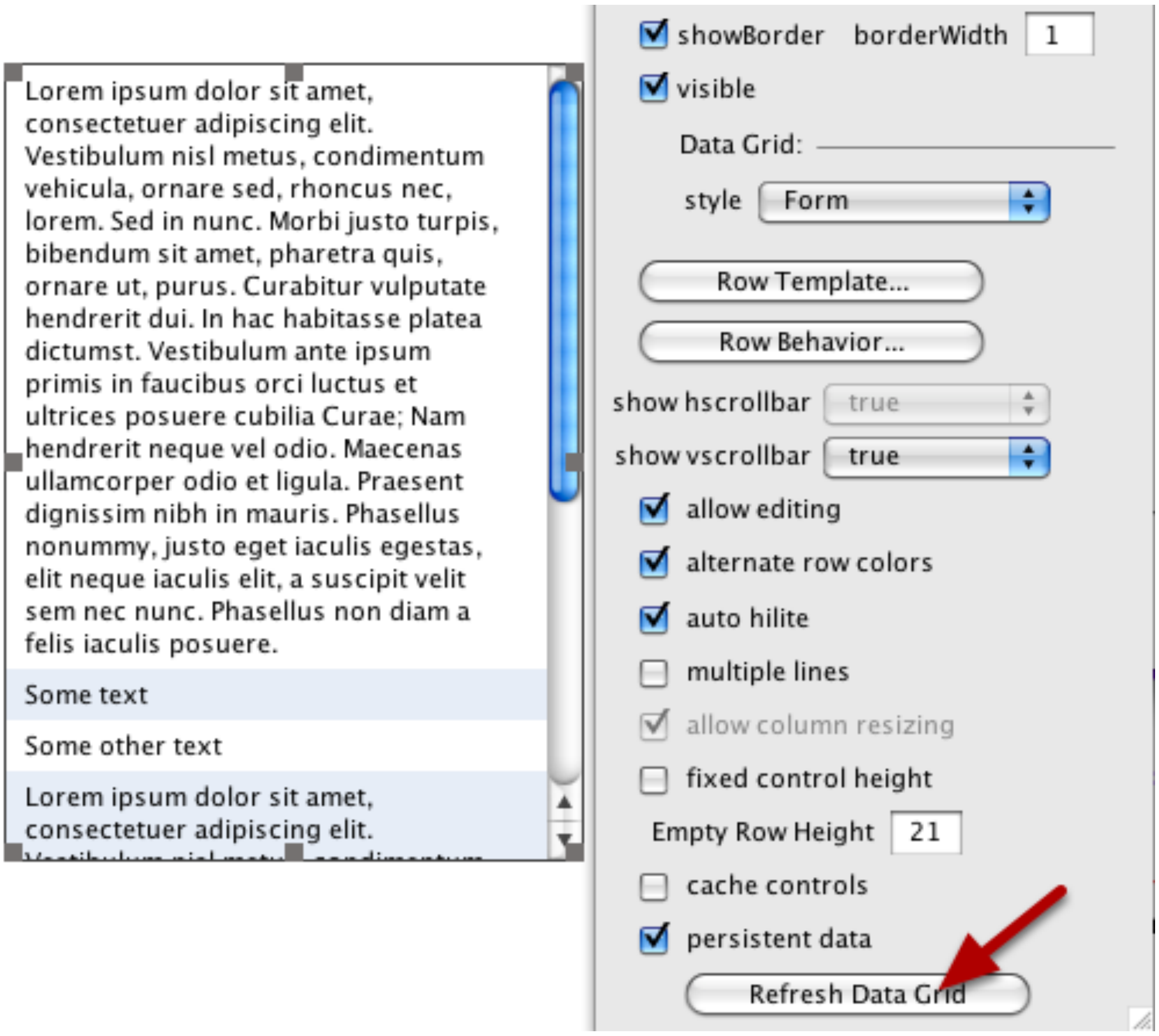


```
on LayoutControl pControlRect
   local theFieldRect

   ## Expand field to fill available width
   put the rect of field "Label" of me into theFieldRect
   put item 3 of pControlRect - 5 into item 3 of theFieldRect
   set the rect of field "Label" of me to theFieldRect

   ## Now resize field to fit content
   put item 2 of theFieldRect \
         + the formattedheight of field "Label" of me - \
         the bottommargin of field "Label" of me \
         into item 4 of theFieldRect
   set the rect of field "Label" of me to theFieldRect

   ## Now update the bounding rect to match total height you
   ## want this row to have
   put item 4 of theFieldRect into item 4 of pControlRect

   set the rect of graphic "Background" of me to pControlRect
end LayoutControl
```

You only need to make a few modifications to the default **LayoutControl** handler in order to get your field to resize to fit the height.

The default LayoutControl handler resizes the field to fill the available width (1). After that is done you then need to resize the field to fill the formattedHeight (2). Finally, resize adjust the rect of the Background graphic to take into account the new field height (3).

## Refresh Data Grid Contents



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Vestibulum nisl metus, condimentum vehicula, ornare sed, rhoncus nec, lorem. Sed in nunc. Morbi justo turpis, bibendum sit amet, pharetra quis, ornare ut, purus. Curabitur vulputate hendrerit dui. In hac habitasse platea dictumst. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nam hendrerit neque vel odio. Maecenas ullamcorper odio et ligula. Praesent dignissim nibh in mauris. Phasellus nonummy, justo eget iaculis egestas, elit neque iaculis elit, a suscipit velit sem nec nunc. Phasellus non diam a felis iaculis posuere.

Some text

Some other text

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

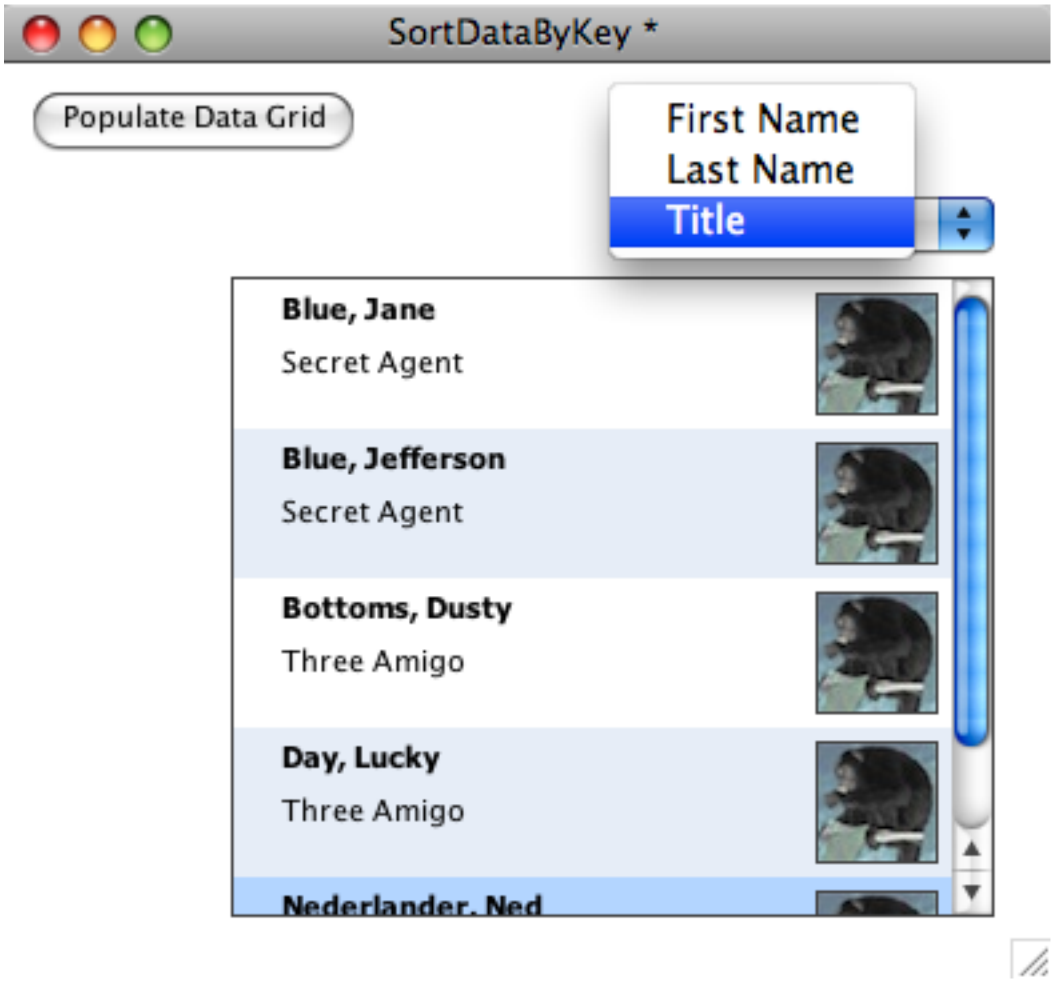
- ✓ showBorder  borderWidth: 1
- ✓ visible
- Data Grid:
- style: Form
- Row Template...
- Row Behavior...
- show hscrollbar: true
- show vscrollbar: true
- ✓ allow editing
- ✓ alternate row colors
- ✓ auto hilite
- ☐ multiple lines
- ✓ allow column resizing
- ☐ fixed control height
- Empty Row Height: 21
- ☐ cache controls
- ✓ persistent data
- Refresh Data Grid

You can now refresh the data grid contents to see the new behavior (1). Notice how each line is resized to fit all of the text.

## How Do I Sort Records By A Specific Key's Values?

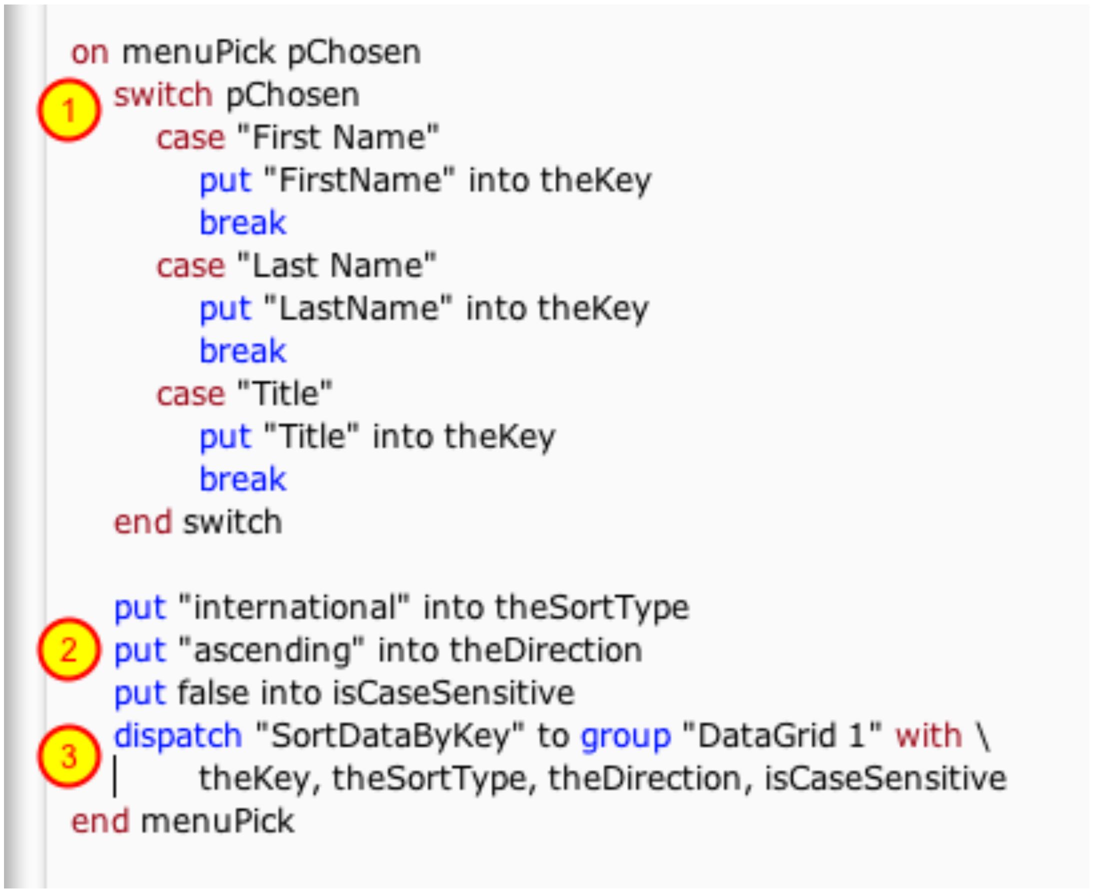
You can sort the rows of a data grid form using the **SortDataByKey** command. Let's look at an example.

### Example



This card has a data grid and an option menu. The option menu contains three values that you can sort by: **First Name**, **Last Name** and **Title**.

## Option Menu Code



```
on menuPick pChosen
   switch pChosen
      case "First Name"
         put "FirstName" into theKey
         break
      case "Last Name"
         put "LastName" into theKey
         break
      case "Title"
         put "Title" into theKey
         break
   end switch

   put "international" into theSortType
   put "ascending" into theDirection
   put false into isCaseSensitive
   dispatch "SortDataByKey" to group "DataGrid 1" with \
         theKey, theSortType, theDirection, isCaseSensitive
end menuPick
```

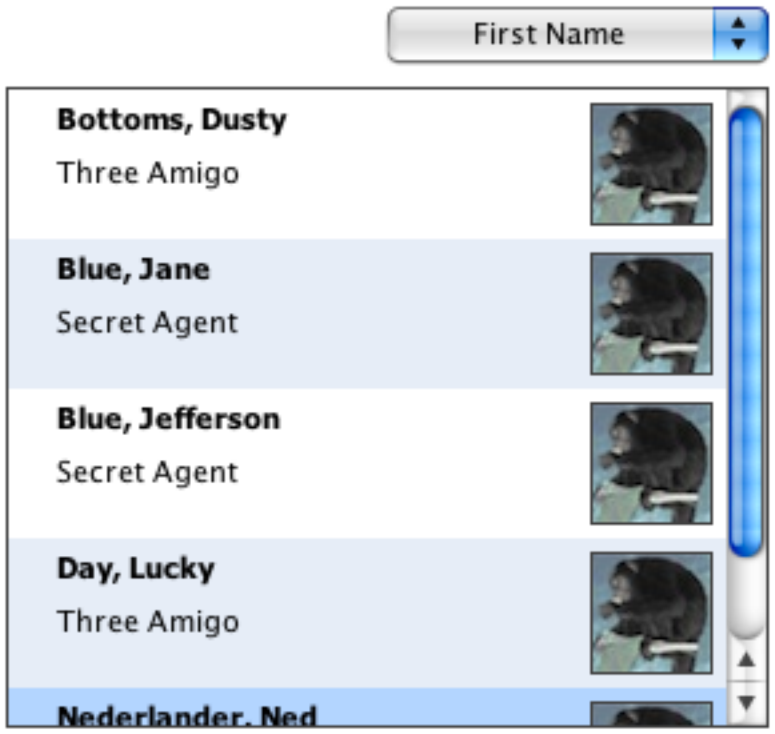
The code to perform the sort is pretty straight forward.

1) Determine which array key of the data grid form to sort by. This will be one the keys you created when you assigned the dgData. If you used the dgText property then the key will be "Label 1" or "Label 2", etc.

2) Determine the sort type, direction and whether or not the sort is case sensitive.

3) Call the SortDataByKey command.

## The Result

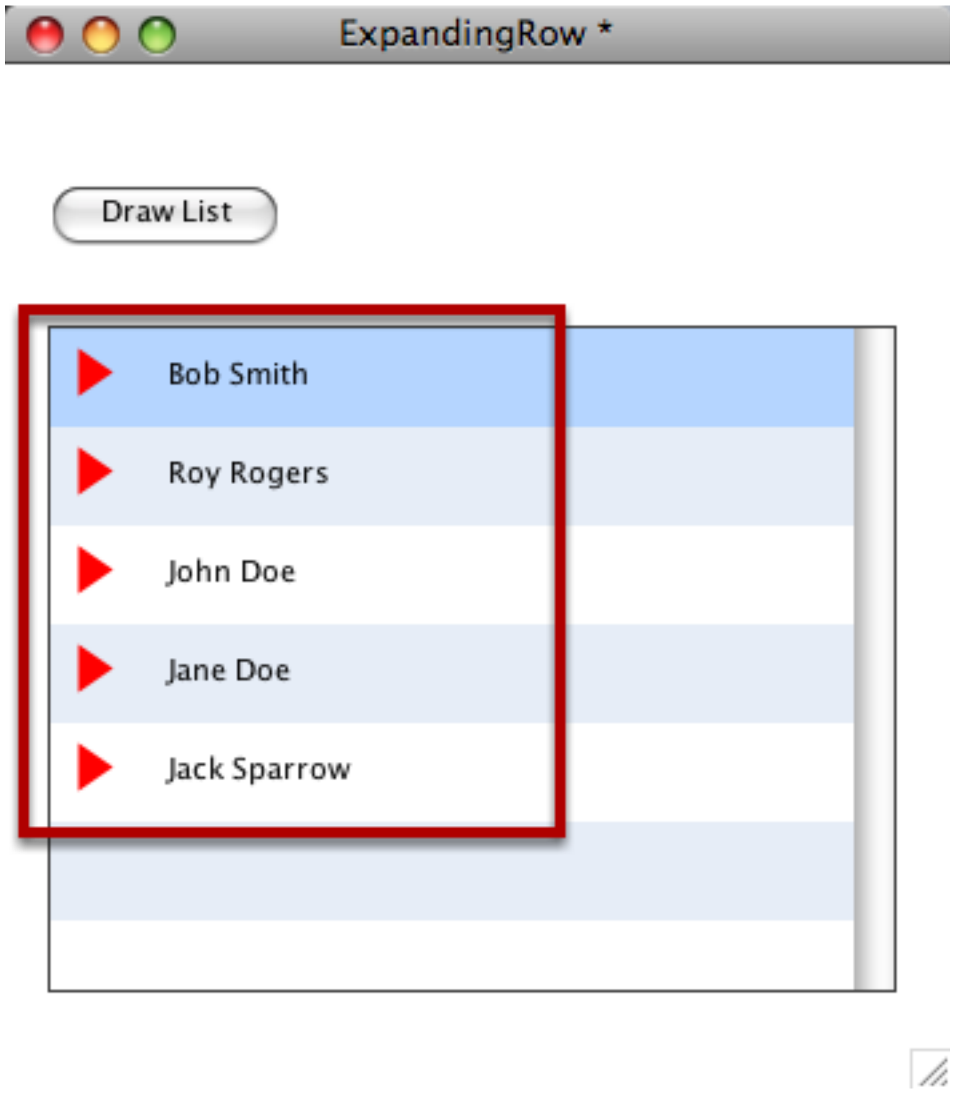


Here is what the sort looks like after selecting First Name.

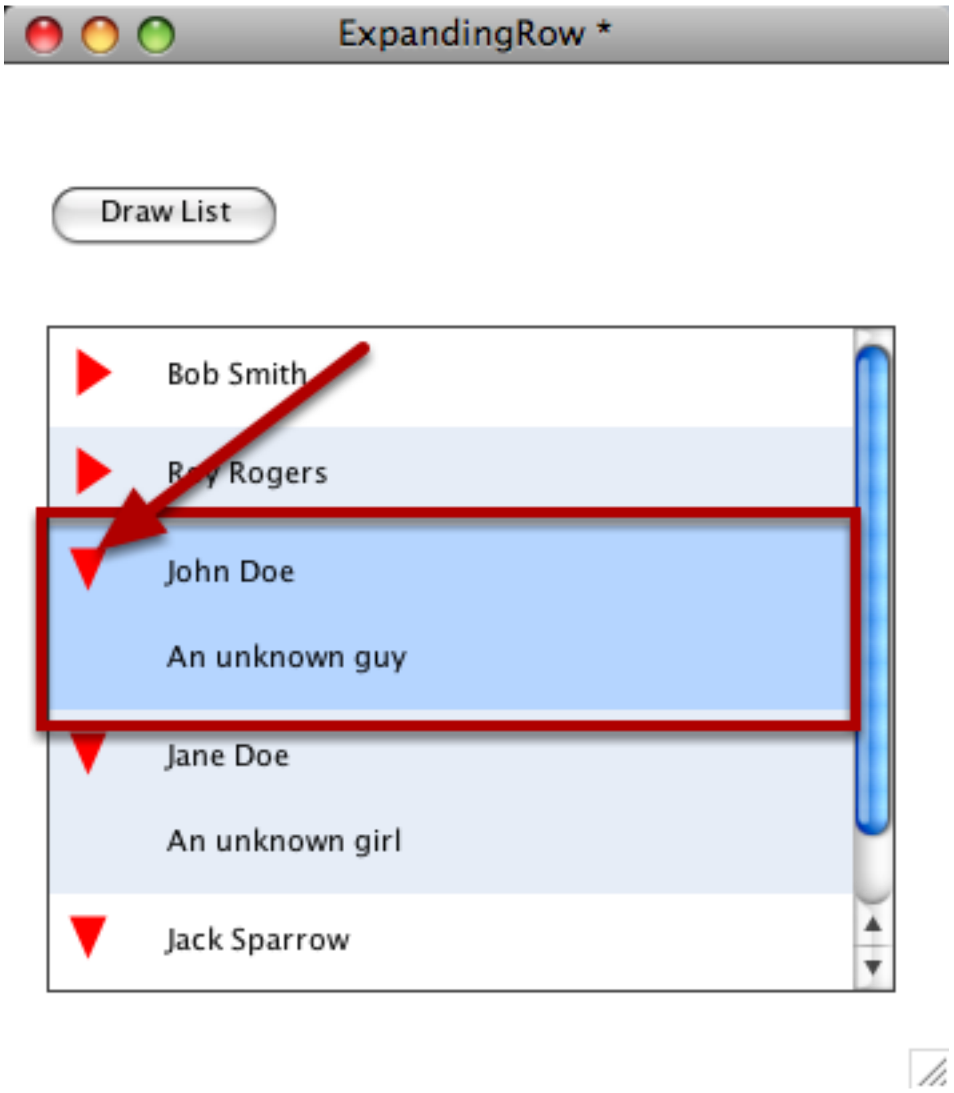
## How Do I Create Rows That Can Expand/Contract?

This lesson will show you how to make a Data Grid form with rows that expand and contract when clicking on an arrow.

### Expanding and Contracting Rows

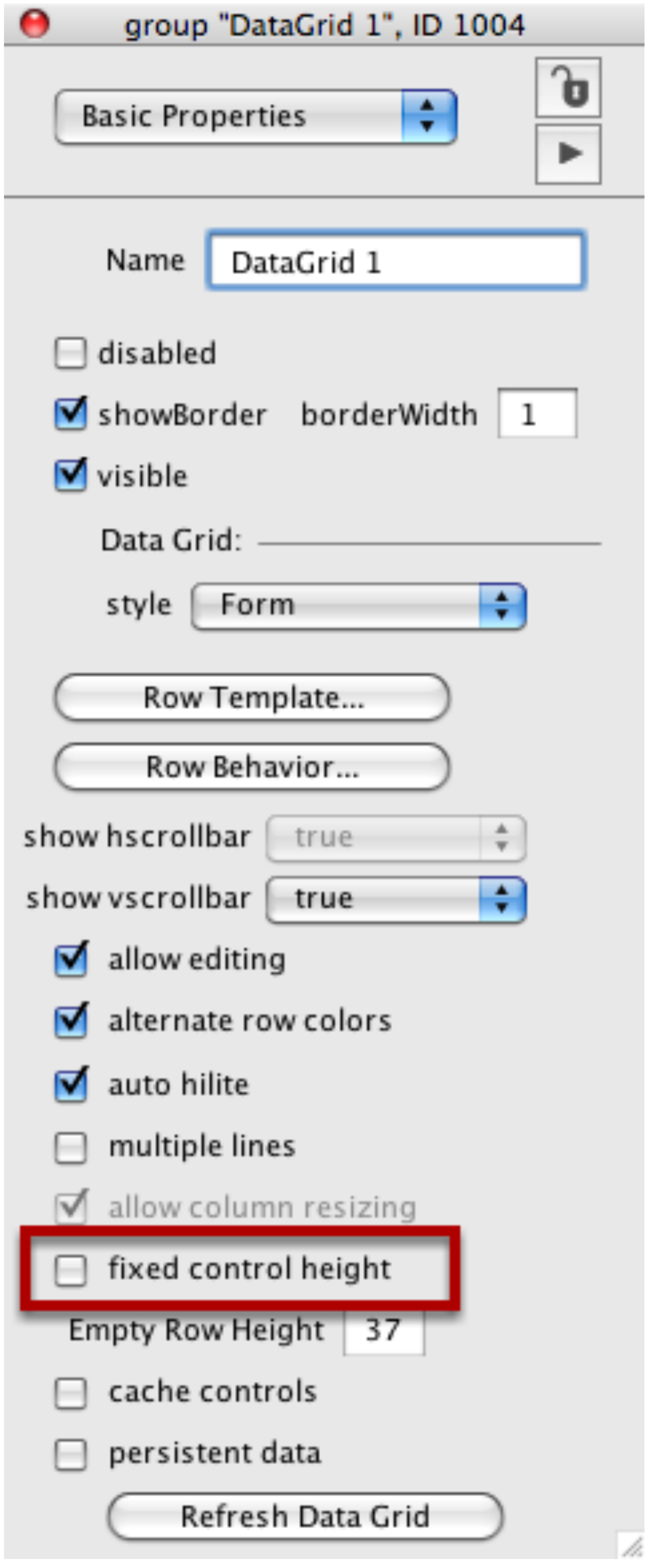


This is what the example stack looks like. By default all of the rows are contracted and only show names.



Clicking on the arrow next to a name expands the row to show the description of the person.

## Setting the Data Grid Properties



group "DataGrid 1", ID 1004

Basic Properties

Name: DataGrid 1

☐ disabled

✓ showBorder  borderWidth: 1

✓ visible

Data Grid:

style: Form

Row Template...

Row Behavior...

show hscrollbar: true

show vscrollbar: true

✓ allow editing

✓ alternate row colors

✓ auto hilite

☐ multiple lines

✓ allow column resizing

☐ fixed control height

Empty Row Height: 37

☐ cache controls

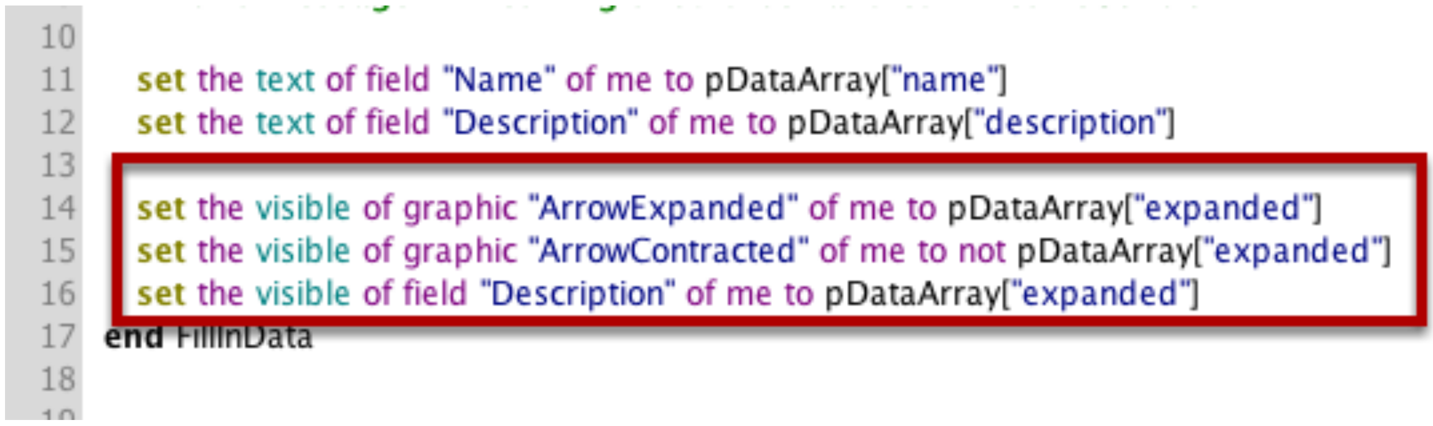
☐ persistent data

Refresh Data Grid

To create a Data Grid form with rows that expand/contact you need to turn off "fixed control height" in the property inspector.

**Note:** To turn off the property using script you set the dgProp["fixed row height"] property to false.

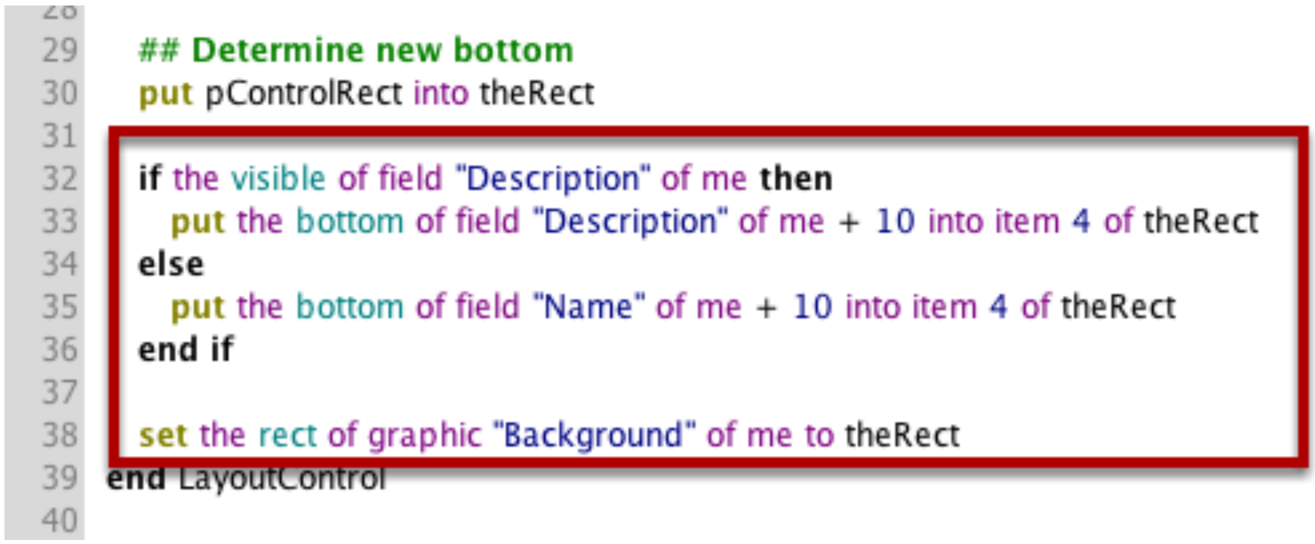
## Updating the Row Behavior Script



```
   set the text of field "Name" of me to pDataArray["name"]
   set the text of field "Description" of me to pDataArray["description"]

   set the visible of graphic "ArrowExpanded" of me to pDataArray["expanded"]
   set the visible of graphic "ArrowContracted" of me to not pDataArray["expanded"]
   set the visible of field "Description" of me to pDataArray["expanded"]
end FillInData
```

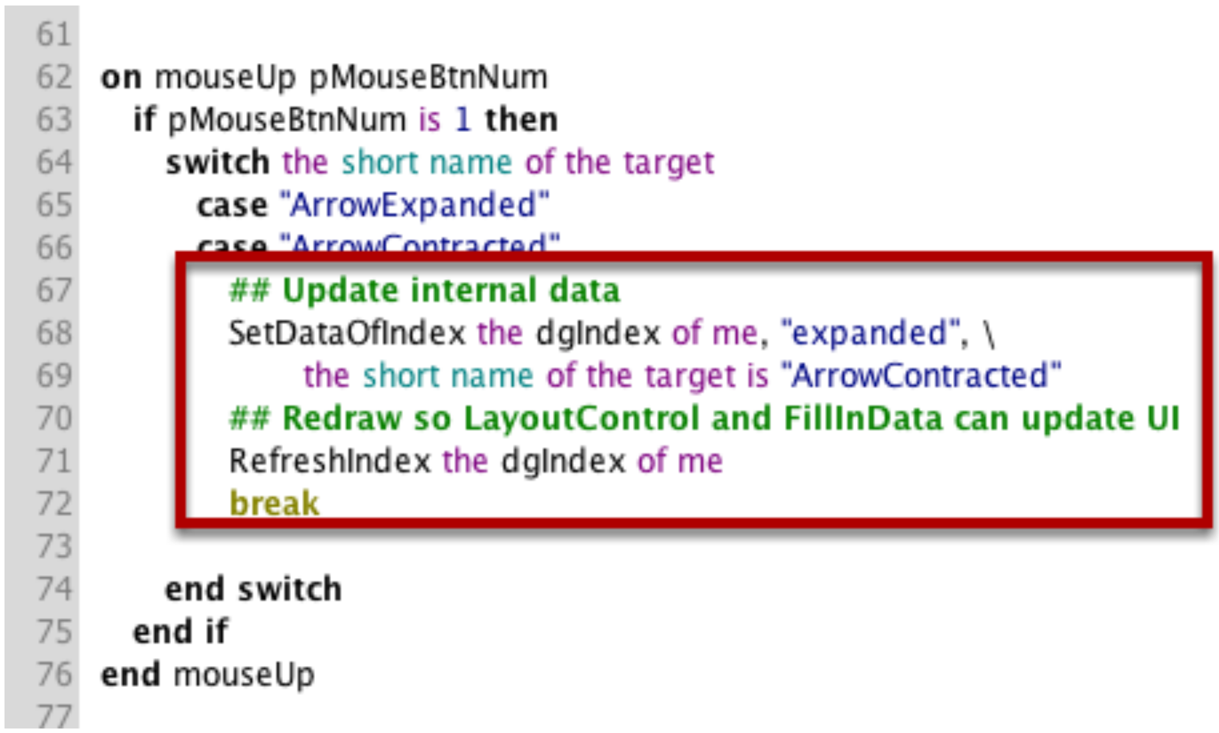
In order to get expanding rows working you need to take a couple of things into account in the **Row Behavior Script**.

In the **FillInData** message you should show or hide controls as necessary. The example stack stores the expanded state of each row in the "expanded" key.



```
   ## Determine new bottom
   put pControlRect into theRect

   if the visible of field "Description" of me then
     put the bottom of field "Description" of me + 10 into item 4 of theRect
   else
     put the bottom of field "Name" of me + 10 into item 4 of theRect
   end if

   set the rect of graphic "Background" of me to theRect
end LayoutControl
```

In the **LayoutControl** message you need to readjust the position of the controls and background graphic based on whether or not the row is expanded or contracted. Since the "fixed row height" property is set to false the Data Grid computes the total height of the data based on the height of each row after calling LayoutControl.



```

on mouseUp pMouseBtnNum
  if pMouseBtnNum is 1 then
    switch the short name of the target
      case "ArrowExpanded"
      case "ArrowContracted"
        ## Update internal data
        SetDataOfIndex the dgIndex of me, "expanded", \
              the short name of the target is "ArrowContracted"
        ## Redraw so LayoutControl and FillInData can update UI
        RefreshIndex the dgIndex of me
        break

    end switch
  end if
end mouseUp

```

When the user clicks on the arrow the example stack updates the **expanded** key in the row's data and then redraws the row. Using SetDataOfIndex in conjunction with RefreshIndex is the most efficient way to do this.

## How Can I Speed Up Drawing When "fixed row height" is False?

When you set the dgProp["fixed row height"] property of a Data Grid to false the Data Grid must draw all records in order to determine the total height. That means that FillInData and LayoutControl are called for every record in the Data Grid. This can be time intensive depending on the amount of data.

This lesson will show you a technique that can speed up the calculation of the total height of the data in situations where the rows can only have heights of known values.

### CalculateFormattedHeight

When the Data Grid loops through the data to calculate the height the message **CalculateFormattedHeight** will be sent to the Row Template. If your Row Behavior handles this message then the Data Grid will use the integer value that you return rather than calling **FillInData** and **LayoutControl**.

Here is an example that uses CalculateFormattedHeight to return the height of a row based on whether or not the row is expanded. Just place the CalculateFormattedHeight message in your Row Behavior script.

```
on CalculateFormattedHeight pDataArray
   if pDataArray["expanded"] then
      return 74
   else
      return 20
   end if
end CalculateFormattedHeight


on FillInData pDataArray
   ...
end FillInData
```

# Using The Built-In Field Editor

## How Do I Open a Table Cell For Editing?

By default a table cell can be edited if the user double-clicks on the cell. This tutorial explains what goes on in the default column behavior so you can customize behavior if you would like.

Note: The default column behavior is stored in button "Default Column" of stack "revDataGridLibrary"

### What You Need to Know

In order to edit the columns of a data grid table you need to know about the following:

1) The EditFieldText command
2) The EditValue message
3) The EditKey and EditKeyOfIndex commands
4) The CloseFieldEditor command which can be sent as a result of calling EditFieldText.

Read up on the entries for EditFieldText, EditValue, EditCell/EditCellOfIndex in the API documentation.

### EditFieldText

The EditFieldText command will create an editor for a field that you specify. The default column behavior calls this command with all three parameters so that data is automatically saved after the user finishes editing.

### EditValue

EditValue is the message that is sent to a row when a request to edit a fields contents has been made. The default column behavior calls EditFieldText when this message is received.

### EditCell and EditCellOfIndex

There are two commands that will open a cell for editing. They are **EditCell** and **EditCellOfIndex**. Each takes the name of the column you want to edit and the line or index you want to edit. Here are two example scripts showing how you could use these commands in the script of a data grid.

```
put "FirstName" into theColumn
put the dgHilitedLine of me into theLineNo
EditCell theColumn, theLineNo
```

```
put "FirstName" into theColumn
put the dgHilitedIndex of me into theIndex
EditCellOfIndex theColumn, theIndex
```

Either of the above calls will trigger the **EditValue** message.

```
on EditValue
   ## Example of opening a field editor for the targeted column.
   ## Since I'm passing in parameters 2 and 3 any changes will automatically be saved to the
dgData.
   EditFieldText the long id of me, the dgIndex of me, the dgColumn of me
end EditValue
```

## CloseFieldEditor

If the user changes any content in the field editor this message will be sent to the field targeted in the first parameter sent to EditFieldText. Read the API docs for EditFieldText which discusses this message.

Example of storing value in dgData in CloseFieldEditor. This would be required if you only passed in one parameter to EditFieldText.

```
on CloseFieldEditor pFieldEditor
   put the dgIndex of me into theIndex
   put the dgDataOfIndex[theIndex] of the dgControl of me into theDataA
   put the text of pFieldEditor into theDataA[the dgColumn of me]
   set the dgDataOfIndex[theIndex] of the dgControl of me to theDataA
end CloseFieldEditor
```

## How Can The User Edit Field Content in a Data Grid Form?

The data grid commands for creating an editor for a particular field in a row template. This lesson will show you how to use them.

### What You Need to Know

In order to edit field contents in a data grid form you need to know about the following:

1) The EditFieldText command
2) The EditValue message
3) The EditKey and EditKeyOfIndex commands

Read up on the entries for EditFieldText, EditValue and EditKey/EditKeyOfIndex in the API documentation.

### EditFieldText

The EditFieldText command will create an editor for a field that you specify. You can use this command to create a field editor for a feld in your row template.

### EditValue

EditValue is the message that is sent to a row when a request to edit a fields contents has been made. You can call EditFieldText from within this message to begin an editing operation.

### EditKey and EditKeyOfIndex

EditKey and EditKeyOfIndex will trigger the EditValue message in a row. Each takes the name of the key you want to edit and the line or index you want to edit. Here are two example scripts showing how you could use these commands in the script of a data grid.

```
## Placed in script of row template behavior
on mouseDown pBtnNum
   if pBtnNum is 1 then
      ## Did the user click on the FirstName field?
      if the short name of the target is "FirstName" then
         put "FirstName" into theKey
         put the dgHilitedLine of me into theLineNo
         EditKey theKey, theLineNo
      end if
   end if
end mouseDown


on mouseDown pBtnNum
   if pBtnNum is 1 then
      ## Did the user click on the FirstName field?
      if the short name of the target is "FirstName" then
         put "FirstName" into theKey
         put the dgHilitedIndex of me into theIndex
         EditKeyOfIndex theKey, theIndex
      end if
   end if
end mouseDown
```

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

```
on EditValue pKey
   ## Example of opening a field editor for the field displaying the value for pKey
   ## Since I'm passing in parameters 2 and 3 any changes will automatically be saved to the dgData.
   ## 'me' is the Data Grid in this case.
   EditFieldText the long id of field pKey of me, the dgHilitedIndex of me, pKey
end EditValue
```

## CloseFieldEditor

If the user changes any content in the field editor this message will be sent to the field targeted in the first parameter sent to EditFieldText. Read the API docs for EditFieldText which discusses this message.

Here is an example of storing the new value in the dgData of the Data Grid in the CloseFieldEditor handler. Manually storing the value would be required if you only passed in one parameter to EditFieldText.

This example script would be in the Row Behavior script as it uses the dgIndex of me property.

```
on CloseFieldEditor pFieldEditor
   ## 'me' is the row control
```

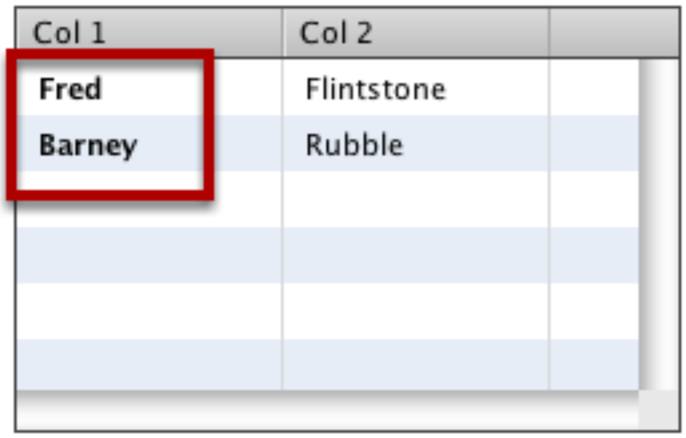
```
    put the dgIndex of me into theIndex
    put the dgDataOfIndex[theIndex] of the dgControl of me into theDataA
    put the text of pFieldEditor into theDataA[the dgColumn] of me
    set the dgDataOfIndex[theIndex] of the dgControl of me to theDataA
end CloseFieldEditor
```

## How Can I Edit The Text as UTF-8, UTF-16 or HTML?

The default Data Grid behavior when editing cell contents is to use the **text** property of the cell as the default value to be edited. This lesson will show you how to provide a specific value for the Data Grid to use as the value to edit.

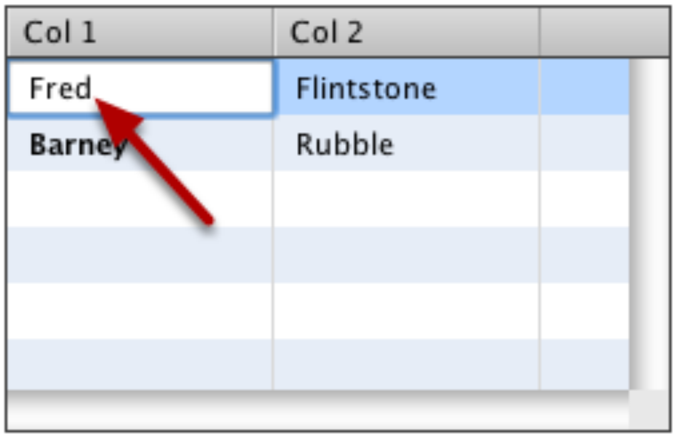
The technique described requires that you create a custom column behavior as outlined in this lesson.

### The Default Behavior



| Col 1 | Col 2 |
|---|---|
| **Fred** | Flintstone |
| **Barney** | Rubble |

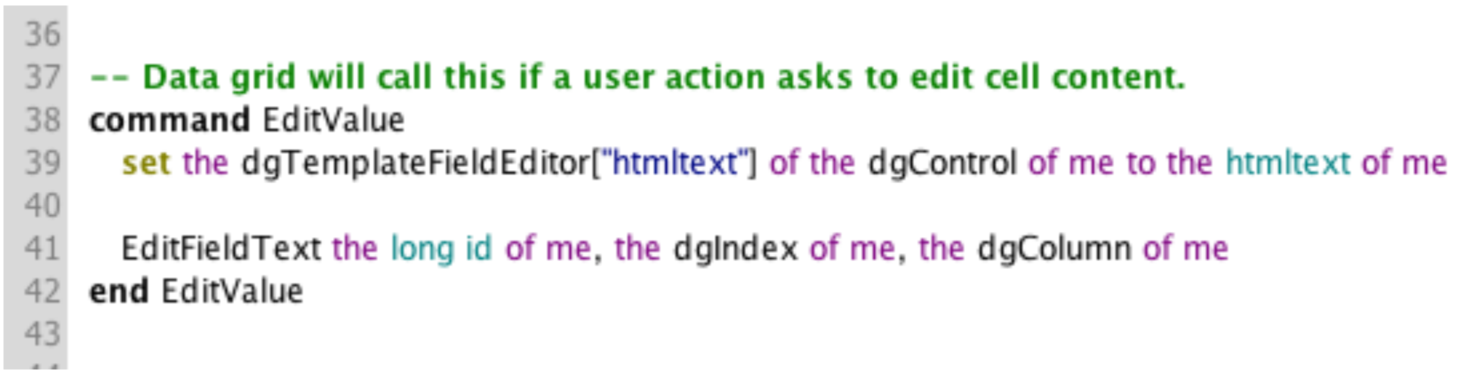
Assume you have a column in your table that displays text with some styling. In this example the name in column 1 is bold.



| Col 1 | Col 2 |
|---|---|
| Fred | Flintstone |
| **Barney** | Rubble |

When you edit the content of the cell the formatting is lost however. This is because the Data Grid edits the text property of the field by default.

## Changing the Default Value For the Editor



```
-- Data grid will call this if a user action asks to edit cell content.
command EditValue
  set the dgTemplateFieldEditor["htmltext"] of the dgControl of me to the htmltext of me

  EditFieldText the long id of me, the dgIndex of me, the dgColumn of me
end EditValue
```

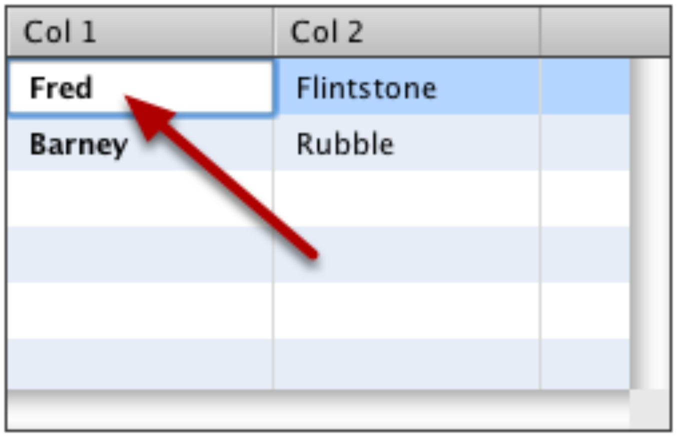
You can change the default value of that the field editor uses by setting a value of the dgTemplateFieldEditor property of the Data Grid. You set the property before calling **EditFieldText**.

For example, if you wanted to display the bold text for editing you could set the dgTemplateFieldEditor["htmltext"] property to the htmltext of the column being edited:

```
set the dgTemplateFieldEditor["htmltext"] of the dgControl of me to the htmltext of me
```

Other properties you can set include "rtftext", "text", "unicodetext" and "utf8text".

## The Result



| Col 1 | Col 2 |
|---|---|
| **Fred** | Flintstone |
| **Barney** | Rubble |

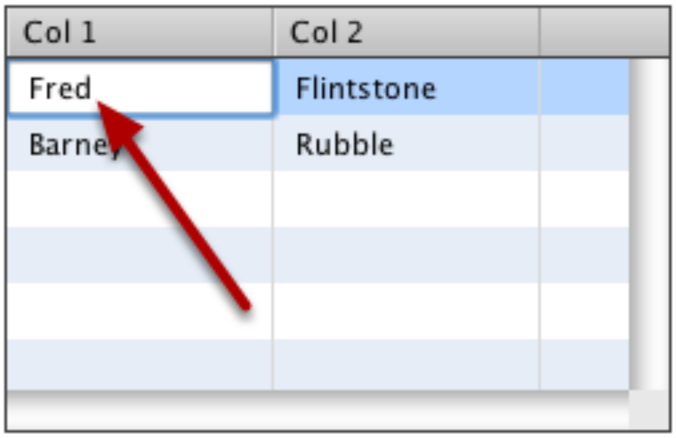
Here is the result of using the example code above in the custom column behavior. Notice how the text being edited is bold.

## How Can I Select The Text in the Edit Field When It Opens?

The default Data Grid behavior when editing cell contents is to put the cursor at the end of the field. This lesson will show you how to tell the Data Grid that all of the cell text should be selected.

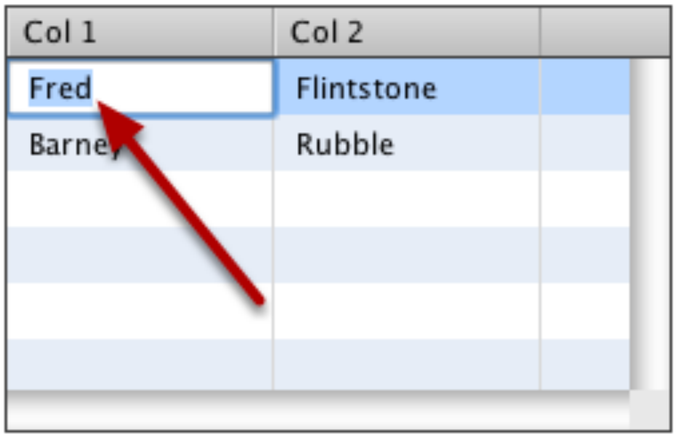
The technique described requires that you create a custom column behavior as outlined in this lesson.

### The Default Behavior



By default the Data Grid will not select the text of a cell when you start editing a value.

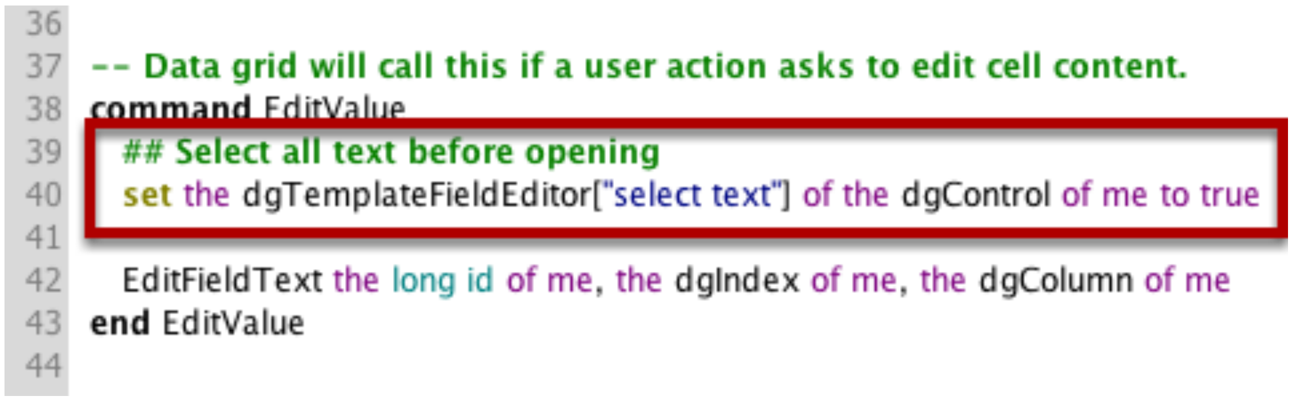
### Selecting The Text



You can tell the Data Grid to select all of the text by setting a special custom property called dgTemplateFieldEditor. If you set the dgTemplateFieldEditor["select text"] of the Data Grid to true then the data Grid will select the cell text.

```
set the dgTemplateFieldEditor["select text"] of the dgControl of me to true
```

## An Example Script



```
-- Data grid will call this if a user action asks to edit cell content.
command EditValue
  ## Select all text before opening
  set the dgTemplateFieldEditor["select text"] of the dgControl of me to true

  EditFieldText the long id of me, the dgIndex of me, the dgColumn of me
end EditValue
```

You can add this line of text to the **EditValue** handler in your custom column behavior. Just place the code right before the call to **EditFieldText**.

```
## Select all text before opening
set the dgTemplateFieldEditor["select text"] of the dgControl of me to true
```

## How Do I Save Changes The User Makes In An Editor Field To An External Data Source?

### When Calling EditFieldText with 3 Parameters (Simpler)

When calling EditFieldText with all three parameters (which is what a data grid column does by default) the data grid will automatically save the text that the user enters in the dgData array. That means that all you need to do is save the text of the editor field to your external data source in the **CloseFieldEditor** message. Note that if you save anything other than the *text of pFieldEditor* then your data source will not match the dgData value.

Here is an example script that goes in the Data Grid group script.

```
## An example that saves
on CloseFieldEditor pFieldEditor
   put the dgColumn of the target into theColumnBeingEdited
   put the text of pFieldEditor into theNewText

   ## Save data to database using command I defined
   put "Person" into theTable
   ## Get the unique id of the row in the database we want to edit
   put GetDataOfIndex(the dgIndex of me, "id") into theRowID

   SaveDataToDatabase theTable, theRowID, theColumnBeingEdited, theNewText
end CloseFieldEditor
```

### When Calling EditFieldText with 1 Parameter (More Flexible)

If you are working with data grid forms or decide to override the default behavior for data grid columns then you will make the call to **EditFieldText** yourself. For complete control over what gets saved you only pass in 1 parameter to **EditFieldText**. By passing in 1 parameter you are responsible for saving any changes to dgData.

Assume you made the following call in a data grid form:

```
EditFieldText the long id of field "Name" of me
```

CloseFieldEditor would still be sent when the user changes the contents of the field but would need to contain code for saving the changes to the dgData array.

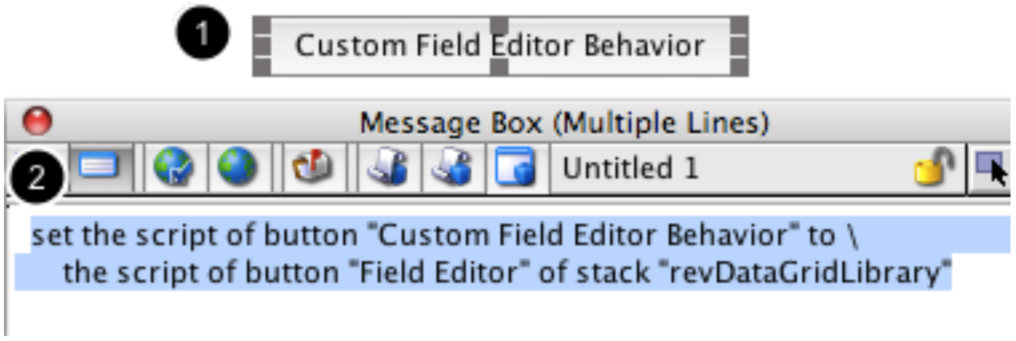
Here is an example script that goes in the Data Grid group script.

```
## CloseFieldEditor placed in data grid script
on CloseFieldEditor pFieldEditor
   put the dgColumn of the target into theColumnBeingEdited
   ## Store UTF8 text
   put unidecode(the unicodetext of pFieldEditor, "UTF8") into theNewText

   ## Save data to database using command I defined
   put "Person" into theTable
   ## Get the unique id of the row in the database we want to edit
   put GetDataOfIndex(the dgIndex of me, "id") into theRowID

   SaveDataToDatabase theTable, theRowID, theColumnBeingEdited, theNewText

   ## Update dgData.
   ## Setting dgDataOfIndex will refresh the data grid display as well as update dgData
   put the dgDataOfIndex[ the dgIndex of the target] of me into theDataA
   put theNewText into theDataA[theColumnBeingEdited]
   set the dgDataOfIndex[the dgIndex of the target] of me to theDataA
end CloseFieldEditor
```

## How Can I 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. If you need data entry to behave differently you can assign your own behavior script to the field editor before it opens. This lesson will show you how.

### Create Your Behavior Script

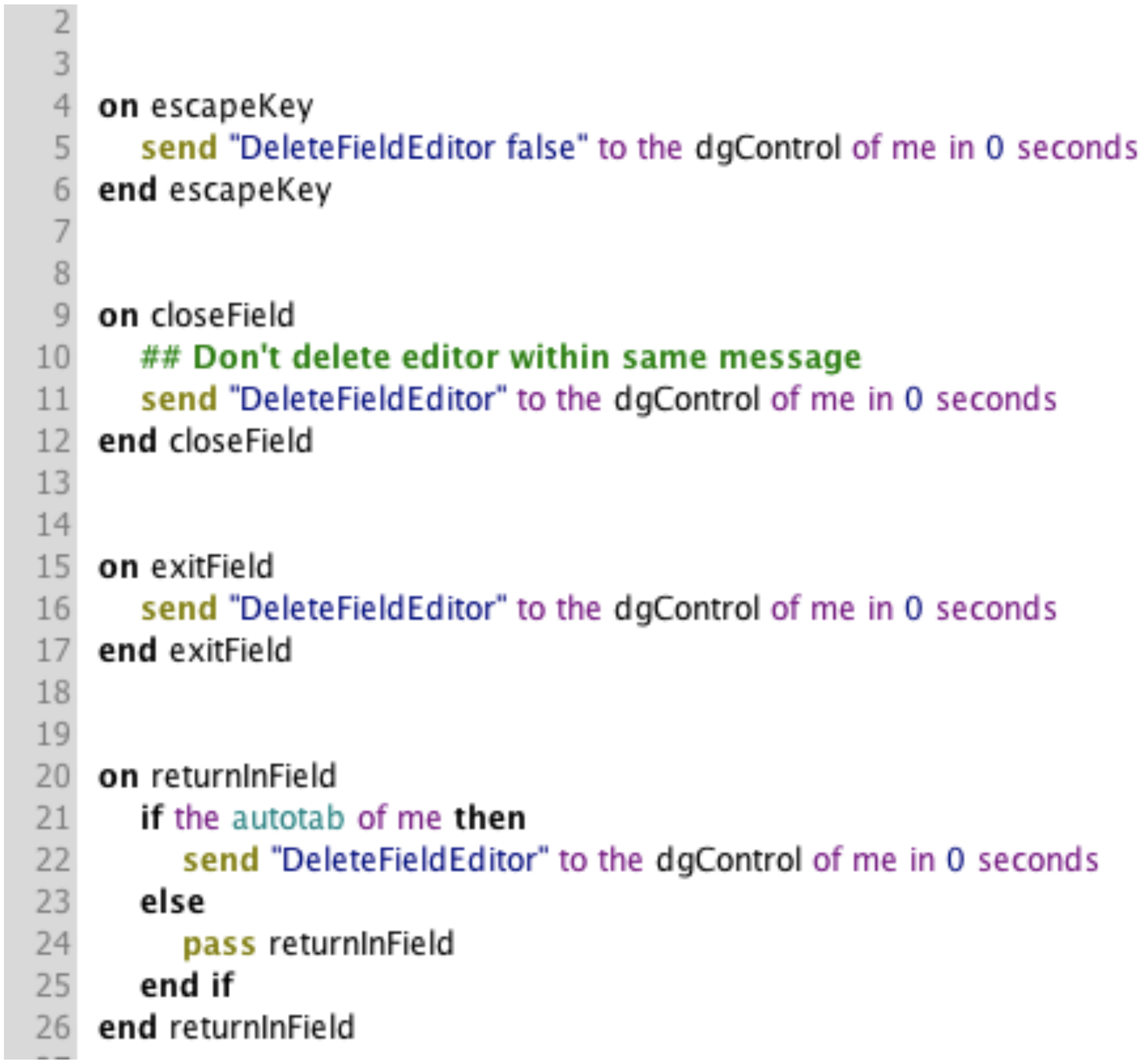


```
set the script of button "Custom Field Editor Behavior" to \
   the script of button "Field Editor" of stack "revDataGridLibrary"
```

1) Create a button to hold the behavior script you want to use with the Data Grid field editor. I've placed this button on the same card as the data grid.

2) Set the script of your button to the script of button "Field Editor" of stack "revDataGridLibrary". This is the behavior script that Data Grid uses by default and is a good place to start when customizing the behavior.

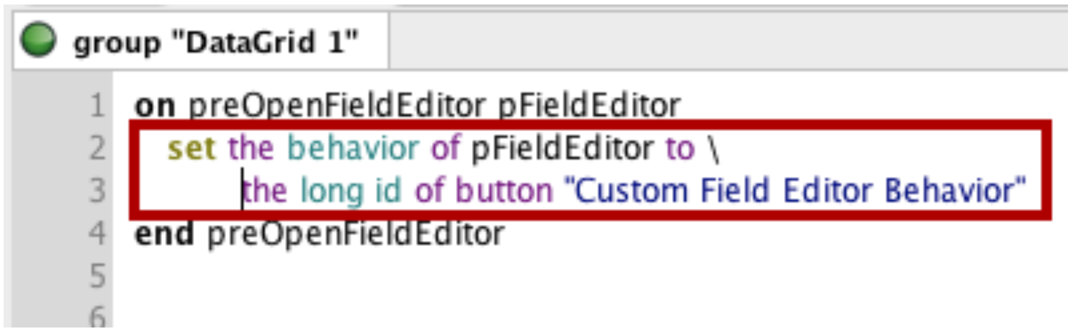
## Customize Your Script



```


on escapeKey
   send "DeleteFieldEditor false" to the dgControl of me in 0 seconds
end escapeKey


on closeField
   ## Don't delete editor within same message
   send "DeleteFieldEditor" to the dgControl of me in 0 seconds
end closeField


on exitField
   send "DeleteFieldEditor" to the dgControl of me in 0 seconds
end exitField


on returnInField
   if the autotab of me then
      send "DeleteFieldEditor" to the dgControl of me in 0 seconds
   else
      pass returnInField
   end if
end returnInField
```

Make any customizations you need to make.

## Assign Your Custom Behavior to Field Editor



group "DataGrid 1"

```
on preOpenFieldEditor pFieldEditor
  set the behavior of pFieldEditor to \
        the long id of button "Custom Field Editor Behavior"
end preOpenFieldEditor
```

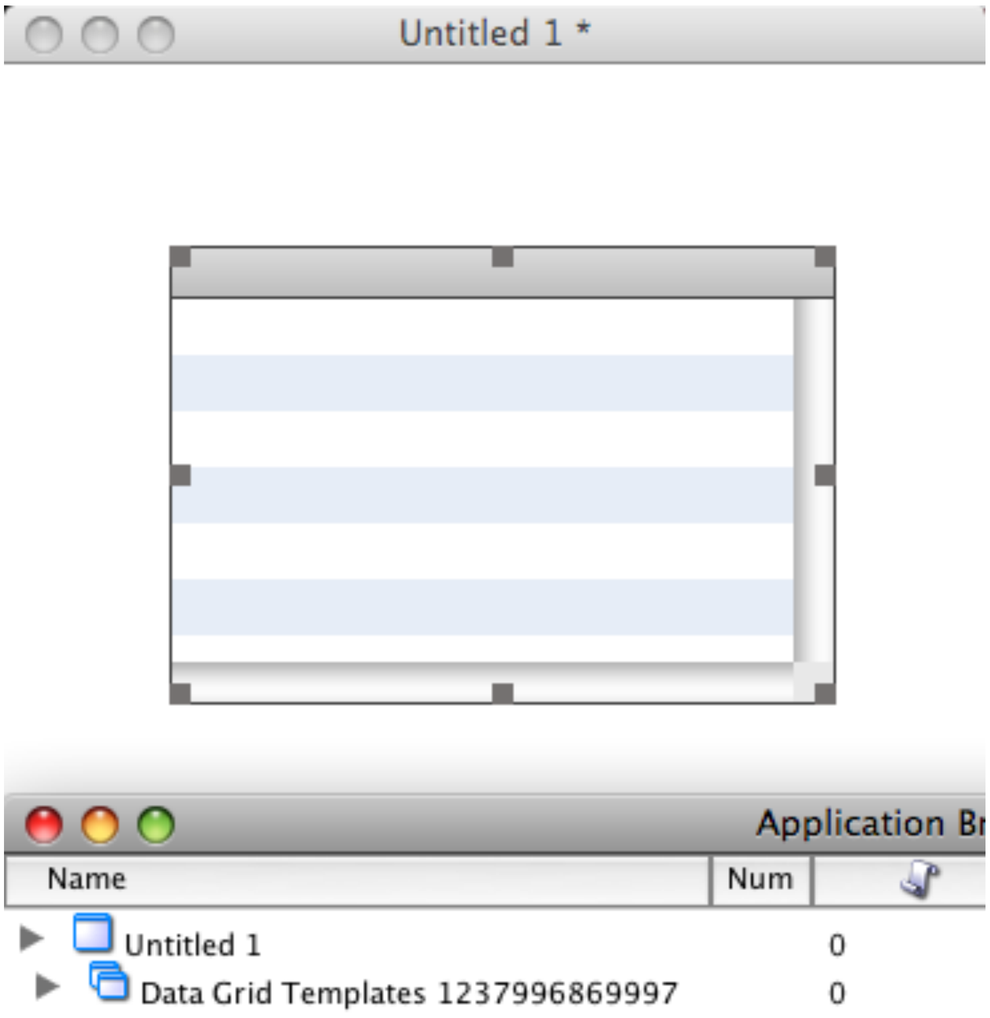
Whenever the Data Grid displays the field editor (e.g. the user double-clicks on a cell in a table) a **preOpenFieldEditor** message is sent the Data Grid. The first parameter is a reference to the field editor control. This is where you can assign your behavior script to the field.

# Building Standalones With The Data Grid

## What Do I Need to Do To Deploy a Standalone With A Data Grid?

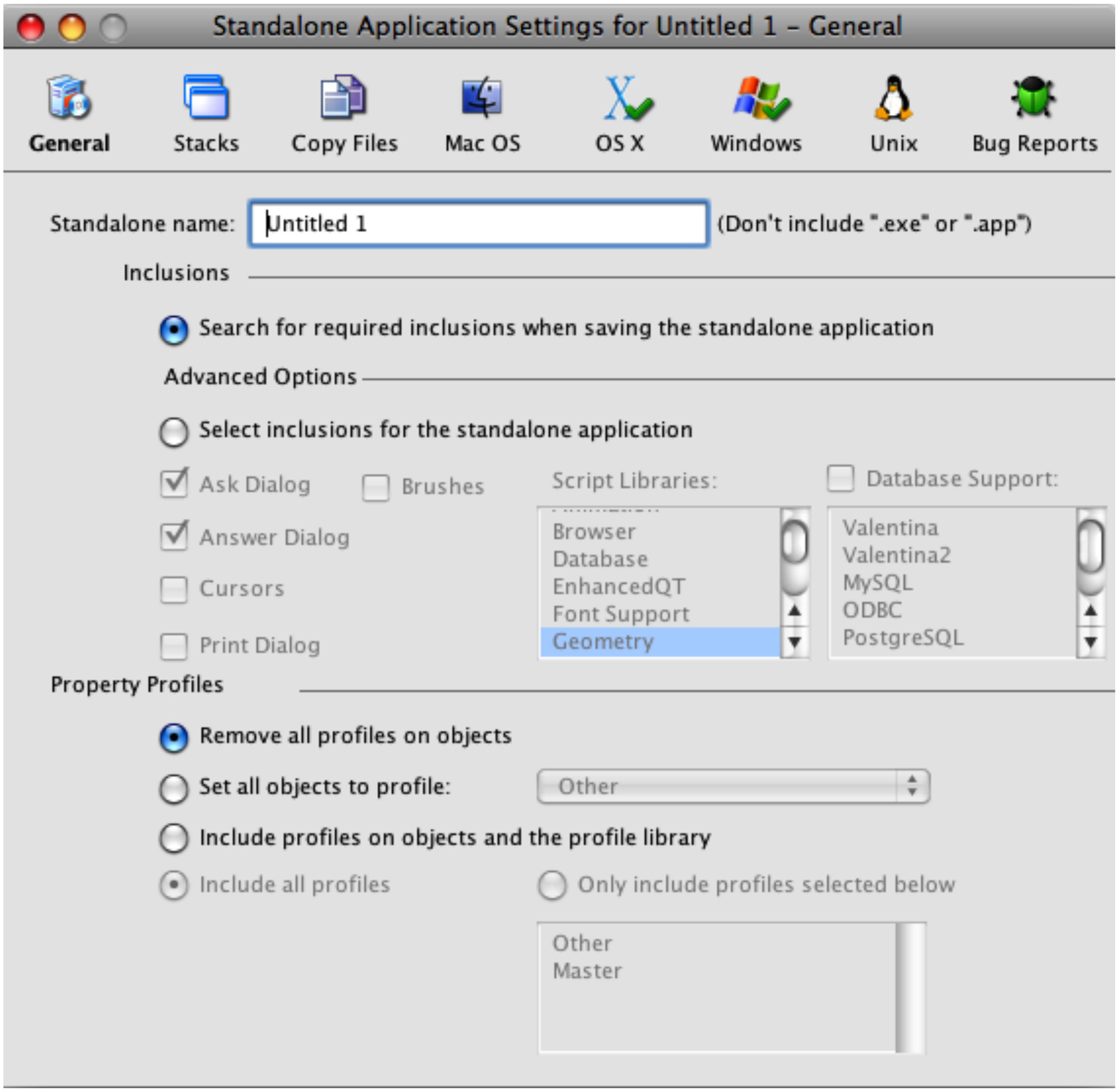
A data grid relies on the stack revDataGridLibrary in order to function properly. This lesson will describe how to include this stack in your standalone applications.

### How The Standalone Builder Adds The Necessary Files



| Name | Num |
|---|---|
| Untitled 1 | 0 |
| Data Grid Templates 1237996869997 | 0 |

When you add a data grid to a stack, a substack is created whose name begins with "Data Grid Templates".



Standalone name: Untitled 1 (Don't include ".exe" or ".app")

Inclusions

[✓] Search for required inclusions when saving the standalone application

Advanced Options

Select inclusions for the standalone application

[✓] Ask Dialog
[ ] Brushes
[✓] Answer Dialog
[ ] Cursors
[ ] Print Dialog

Script Libraries: Browser, Database, EnhancedQT, Font Support, Geometry

[ ] Database Support: Valentina, Valentina2, MySQL, ODBC, PostgreSQL

Property Profiles

[✓] Remove all profiles on objects

Set all objects to profile: Other

Include profiles on objects and the profile library

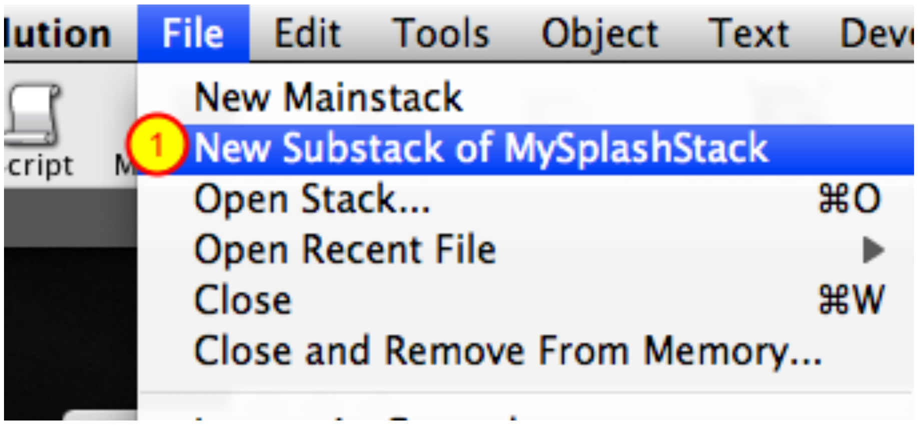
Include all profiles

Only include profiles selected below

Other
Master

When you build a standalone application Revolution looks for a substack whose name begins with "Data Grid Templates". If it finds one then the revDataGridLibrary stack is added to your standalone application.

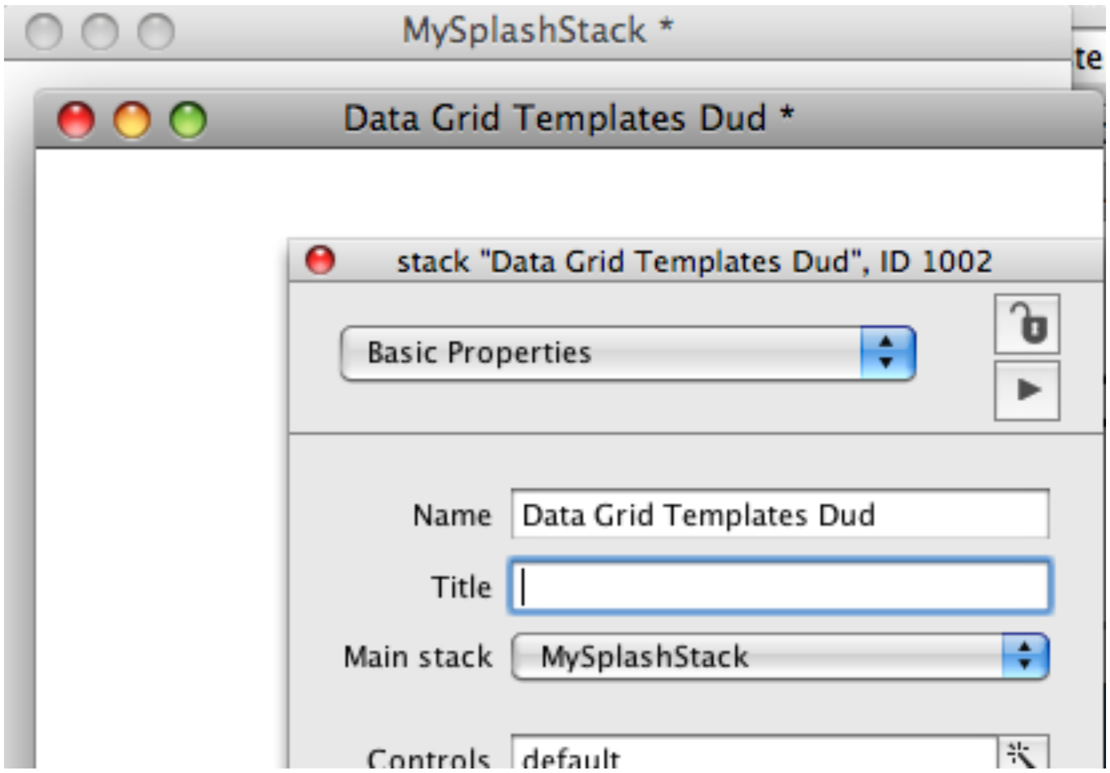
## But What About Launcher Stacks (Splash Stacks)?



Some developers prefer to use a launcher (or splash) stack technique. This technique builds a standalone using a stack with very little code in it. In this case the stack used to build the standalone will not have a substack whose name begins with "Data Grid Templates" as no data grids have been added to it.

In this case you will need to create a substack that tricks the Standalone Builder into adding the necessary stack files. From the File menu, create a new substack of your launcher stack (1).

## Name the Substack


MySplashStack *
Data Grid Templates Dud *
stack "Data Grid Templates Dud", ID 1002
Basic Properties

Name: Data Grid Templates Dud
Title:
Main stack: MySplashStack
Controls: default

Name the substack "Data Grid Templates Dud" and save your launcher stack. Now if you build a standalone application the Standalone Builder will include the necessary data grid stack.

# Useful Things To Know

## What Sorts of Things Should I Not Do In Order To Avoid Needless Suffering?

There are some things that you could do that will cause you to scratch your head when things go wrong. Since knowing is half the battle we will share the issues we are aware of.

### Don't call a handler that redraws the data grid from within a control in the data grid

This will generate an error since you are deleting a control that is currently executing code. This is a no-no and the Revolution engine will complain and stop executing. You can avoid calling a handler that refreshes the data grid from within a control by a) using send in time or b) placing the code in the data grid script itself.

Example A

```
## Script that is in the row behavior for a data grid.
send "DeleteIndex theIndex" to the dgControl of me in 0 seconds
```

Example B

```
## Script that deletes hilited index on mouseUp.
## Place in the data grid script.
on mouseUp pMouseBtnNum
   if pMouseBtnNum is 1 then
      put the dgHilitedIndex of me into theIndex
      DeleteIndex theIndex
   end if
end mouseUp
```

### Don't try to draw a Data Grid on a card that Is 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 Revolution engine unless the field is on an open card.

### Do not lock messages when accessing data grid properties.

If messages are locked when you try to access a data grid property (i.e. the dgProps["alternate row color"] of group "DataGrid") then the correct value will not be returned/set. When messages are locked getProp/setProp handlers are not triggered in Revolution and the data grid relies on these.

### Do not password protect the Data Grid Templates Stack

The data grid copies the templates from the **Data Grid Templates xxxx** stack. If you password protect this stack then the data grid will be unable to copy the templates.

## Don't Rename the "Data Grid Templates" Stack

If you rename this stack then all of your data grids with templates stored in the stack will stop working. Since the data grid can no longer locate the custom templates they will fail to draw properly.

## Don't Try to Search When Data is Being Loaded From an External Source

## Stop Editing the Template Group Before Drawing Your Data Grid

When you edit a group in Revolution the engine no longer knows that the group exists. If you try to draw a data grid while editing it's template group then the data grid will fail to draw.

# Advanced Options

## 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 your data. But what about situations where creating an array is too time intensive and you already have the data in another format? A data grid can handle these situations for you as well using a feature called callbacks.

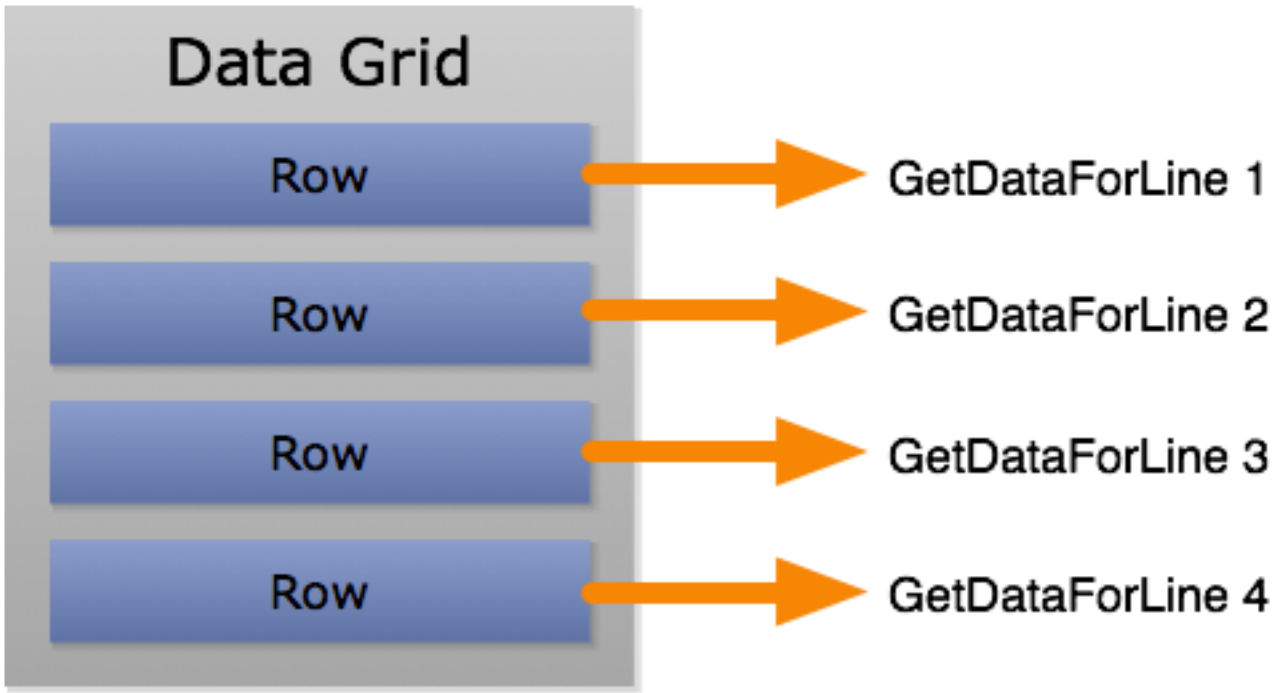
Click here to download a sample stack that shows how to use the techniques described in this lesson to display data from a SQLite database.

### 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



This diagram demonstrates how this works. If you were to set the dgNumberOfRecords to 10000 then 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 you define **GetDataForLine** as follows:

```
command GetDataForLine pLine, @pDataA

end GetDataForLine
```

You can define this handler in the data grid script or anywhere else in the message path. Just fill in pDataA with the appropriate data and the data grid will display it.

## Creating A Data Grid By Hand

This lesson will show you how to create a data grid through script.

### 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 theDataGridRef
```

### Set The "style" Property

```
set the dgProp["style"] of theDataGridRef to "table" or "form"
```

### Assign a Row Template

```
set the dgProp["row template"] of theDataGridRef to the long id of group "MyRowTemplate" of stack "MyStack"
```

You should probably create this ahead of time using the IDE. For example, you could 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).

# API and Properties

## Data Grid Properties

### General Properties

All general data grid properties are stored in the dgProps custom property set. To access a property you can use dgProps or dgProp:

```
put the dgProp[ "PROPERTY_NAME" ] of group "Data Grid"
```

*allow editing*
- Set to true to allow users to edit the cells of a table or the fields in a form. Note that when customizing templates for form rows or table columns you would check this property to determine if fields should allow editing. The default Row Behavior script contains an example of how to do this.

*alternate row color*
- The color of every other row's background. Default value is *empty* in which case a default hilite color is used. Only applicable when 'alternate row colors' is true. Prefix property name with "effective" to get the color being used.

*alternate row colors*
- Set to true to alternate the background colors of every other row.

*auto hilite*
- Set to true if you would like the data grid to automatically handle row highlighting in response to user interaction.

*background color*
- The background color of the data grid. Note that if 'alternate row colors' is true then the alternating row colors will cover the background.

*cache controls*
- By default the data grid only draws the controls that are visible on the screen. In circumstances where you don't have large amounts of records but the records you have take a long time to draw you may cache all of the controls when the data grid is drawn. This takes a little longer to display at the beginning but will offer smooth scrolling when the user interacts with the data grid.

*column divider color*
- Sets the color of the table column dividers. Prefix property name with "effective" to get the color being used when this property is set to empty

*column margins*

- Specifies the margins to be applied to each cell in a column.

*control type*
- Returns "data grid". You can check this property to determine if a group is a data grid.

*dim on focusOut*
- If true then the highlighted lines will be dimmed to 'dimmed hilite color' when the data grid does not have focus. Default is true.

*dimmed hilite color*
- Color that highlighted lines will be when control is not focused. Prefix property name with "effective" to get the color being used when this property is set to empty.

*fixed row height*
- Set to true if all of your data will be drawn at the same height. Setting this to true will dramatically improve performance the first time the data grid is drawn as the data grid does not have to determine the height of all of your records before drawing. Default value is true.

*hilite color*
- The color to use when highlighting a row. If empty then *the hiliteColor* property is used. Prefix property name with "effective" to get the color being used when this property is set to empty

*hilited text color*
- The color to apply to text when a row is highlighted. By default this property is empty in which case the color is black if the average of the RGB value for the *hilite color* > 128, white otherwise.

*multiple lines*
- Set to true to allow the user to select multiple lines in the data grid.

*opaque*
- Shows or hides the data grid background.

*persistent data*
- Set to true if you would like the data grid to store the data being displayed between sessions. The data grid always works with data stored in a script local variable but if this value is true then the data will be cached in a custom property as well. This will double the memory used so this is suitable for small lists. For large data sets you should set this property to 'false' and set the data grid data each time the data grid is opened. The default value is 'true'.

*row color*
- The primary row color. This color alternates with 'alternate row color'. Prefix property name with

"effective" to get the color being used when this property is set to empty

*row height*
- For tables and forms whose 'fixed row height' property is true this represents the height that your rows will be drawn at. If this property is not set for a form whose 'fixed control height' is true then the record template group height is used. For forms whose 'fixed control height' is false this represents the height that alternating rows that contain no data will be drawn at.

*row template*
- This is the group that represents a record in your data grid. If the style of the data grid is form then this group will be copied into the data grid. If the data grid style is table then the data grid looks in this group for controls named after the columns in your table. If the data grid finds a control in this group that matches a column name then the control will be used to visually represent the column. Otherwise a field is used.

By default the Revolution IDE creates this group on a card in a stack whose name starts with "Data Grid Template". Clicking the "Row Template" button in the Property Inspector reveal the card containing this group.

*scroll when hscrollbar is hidden*
*scroll when vscrollbar is hidden*
- By default a data grid will not respond to the mouse scroll wheel, page up, page down, home or end if the scrollbar is hidden. Set this property to true if you would like to override this behavior. This is useful if you want to create custom scrollbars.

*scrollbar corner offset*
- This property is an integer that specifies the distance from the corner of the window that the vertical and horizontal scrollbars position themselves when only one of them is visible. This is primarily useful on OS X when your data grid reaches all the way to the bottom right corner of the window where the window drag handle appears. Setting this to a value like "15" will keep your scrollbar controls from being hidden behind the window drag handle. Default value is "0".

*show vscrollbar*
- Toggles the visibility of the vertical scrollbar. True, false, or auto.

*show hscrollbar*
- Toggles the visibility of the horizontal scrollbar for a table. True, false or auto.

*scrollbar width*
- Set to an integer or to *auto* if you would like the data grid to set the appropriate width based on the platform it is being displayed on. Default is *auto*. When set to *auto* you can retrieve the actual width in

pixels using *effective scrollbar width*.

*style*
- 'form' or 'table'.

*text color*
- The text color to apply to the data area of a table or form. Prefix property name with "effective" to get the color being used when this property is set to empty

*text font*
- The font to apply to the data area of a table or form. Prefix property name with "effective" to get the font being used when this property is set to empty.

*text size*
- The text size to apply to the data area of a table or form. Prefix property name with "effective" to get the size being used when this property is set to empty

*text style*
- The text style to apply to the data area of a table or form. Prefix property name with "effective" to get the style being used when this property is set to empty

## Table Properties

All data grid table properties are stored in the dgProps custom property set. To access a property you can use dgProps or dgProp:

```
put the dgProp[ PROPERTY_NAME ] of group "Data Grid"
```

*allow column resizing*
- If true then the user can resize columns in the table header. Note that you can also turn off resizing for individual columns which would override this setting.

*column divider color*
- The color of the column dividers in a table.

*column alignments*
- Allows you to set all column alignment values at once. Line delimited list of alignment values.

*column visibility*
- Allows you to set the visible property for all columns at once. Line delimited list of boolean values.

*column widths*
- Allows you to set all column widths at once. Comma delimited list of integers.

*columns*
- Line delimited list of columns in your table.

*column labels*
- Line delimited list of labels for columns in your table.

*corner color*
- The color of the corner piece that appears when both horizontal and vertical scrollbars are visible. You set set to a solid color, a gradient (two colors, one per line) or an array containing the keys of the fillGradient property.

*default column behavior*
- The table style has an internal behavior that is used for columns which have no custom control defined for them. This behavior sets the text of a field as well as the alignment, etc. You can set this property to a button containing the default behavior you would like. This can be useful if you need to display html, unicode or rtf text among other things. The default value is empty.

To see the script that the data grid uses by default for columns you can edit the script of button "Default Column" of group "Behaviors" of stack "revDataGridLibrary"

*default header behavior*
- The table style has an internal behavior that is used for column headers. If you would like to override the default behavior for column headers you can set this property to point to a button with your own behavior script.

To see the script that the data grid uses by default you can edit the script of button "Default Header" of group "Behaviors" of stack "revDataGridLibrary"

*header background color*
- The background color of the header. You set set to a solid color, a gradient (two colors, one per line) or an array containing the keys of the fillGradient property.

*header background hilite color*
- The background color of the header that is being sorted by. You set set to a solid color, a gradient (two colors, one per line) or an array containing the keys of the fillGradient property.

*header height*
- The height of the header are of the table.

*header margins*
- Specifies the margins to be applied to the fields that display the header text.

*header text color*
- The text color to apply to the header area of a table. Prefix property name with "effective" to get the color being used when this property is set to empty

*header text font*
- The font to apply to the header area of a table. Prefix property name with "effective" to get the font being used when this property is set to empty

*header text size*
- The text size to apply to the header area of a table. Prefix property name with "effective" to get the size being used when this property is set to empty

*header text style*
- The text style to apply to the header area of a table. Prefix property name with "effective" to get the style being used when this property is set to empty

*show column dividers*
- Toggles the visibility of the column dividers in the data display area. Default value is true.

*show column dividers*
- Toggles the visibility of the column dividers that appear in the data area.

*show header*
- Toggles the visibility of the header. Default value is true.

*sort by column*
- The column that the table data is currently being sorted by. You can set this property to sort by a new column.

*visible columns*
- Line delimited list of columns in your table that are visible.

## Column Properties

These properties allow you to set properties of individual columns in a table. The syntax you use resembles:

set the *dg*ColumnSortType [ COLUMN ] of group "Data Grid" to "numeric" where COLUMN is the name of the column you are targeting.

*dgColumnAlignment [COLUMN]*
- Get/set the alignment for a column. Valid values are 'left', 'right' or 'center'.

*dgColumnIsEditable [COLUMN]*
- Toggle whether or not a column is editable by the user. Set to true/false.

*dgColumnIsVisible [COLUMN]*
- Get/set the visibility of the column.

*dgColumnIsResizable [COLUMN]*
- Get/set whether or not a column is resizable.

*dgColumnLabel [COLUMN]*
- Get/set the label used for the column. If the label is empty then the column name is used.

*dgColumnMaxWidth [COLUMN]*
- Get/Set the maximum width that a column can be resized to.

*dgColumnMinWidth [COLUMN]*
- Get/Set the minimum width that a column can be resized to.

*dgColumnName [COLUMN]  pNewName*
- Set a new name for a column.

*dgColumnTemplate [COLUMN]*
- Get the control that is used to visually represent the column in the table. This control will be a control in the 'row template' group with the same name as the column. If no matching control exists then a field is used.

*dgColumnTooltip [COLUMN]*
- Set the tooltip that appears when the mouse is over the column header.

*dgHeaderTemplate [COLUMN]*
- Get the control that is used to visually represent the header for a column. This control will be a control in the 'row template' group name "COLUMN [Header]". If no matching control exists then the deafult header control is used.

*dgColumnSortDirection [COLUMN]*
- Get/set the direction of the sort for the column. Valid values are 'ascending' or 'descending'.

*dgColumnSortIsCaseSensitive [COLUMN]*
- Get/set whether or not column sort is case sensitive. Default value is false.

*dgColumnSortType [COLUMN]*
- Get/set the sort type of the column to 'text', 'numeric', 'datetime' or 'system datetime'.

*dgColumnWidth [COLUMN]*
- Get/set the width of a column.

*dgHeaderAlignment [COLUMN]*
- Get/set the alignment for a column's header.

## Template Field Editor Properties

The template field editor properties are set using the dgTemplateFieldEditor custom property.

set the dgTemplateFieldEditor[ PROPERTY_NAME ] of group "Data Grid" to SOME_VALUE

*select text*
- Set to true to select all text in the field editor.

*text*
- Set to a string that will be assigned to the *text* property of the field editor.

*htmltext*
- Set to a string that will be assigned to the *htmltext* property of the field editor.

*rtftext*
- Set to a string that will be assigned to the *rtftext* property of the field editor.

*unicodetext*
- Set to a string that will be assigned to the *unicodetext* property of the field editor.

*utf8text*
- Set to a string that will be assigned to the *unicodetext* property of the field editor after being converted from UTF-8 to UTF16.

## Table Header Properties

*dgHeader*
- Returns the long id of the group that contains the controls for the table header. Use this property in a mouseDown/mouseUp handler to determine if the user clicked on a the table header. If the dgHeader

of the target is not empty then ... (user clicked in table header).

*dgHeaderControl*
- Returns the long id of the group that contains the controls for a column header. Use this property in a mouseDown/mouseUp handler to determine if the user clicked on a column header. If the dgHeaderControl of the target is not empty then... (user clicked in a column header).

## Custom Properties

*dgControl*
- get the dgControl of the target
- Returns the long id of the data grid. Useful in row/column template behaviors when you need to get properties of the data grid.

*dgData*
- get the dgData
- set the dgData of group "DataGrid" to pDataArray
- Get or set the data array that the data grid will display. The first dimension of the array uses numeric keys and the value of each is an array. You can store anything you would like in each numeric key's array. For data grid tables the keys should match the column names in order for the data grid to correctly map the array value to the column cell. The following array would represent two records in the data grid:

```
put "Hi" into theA[1]["message"]
put "Bye" into theA[2]["message"]

set the dgData of group "DataGrid" to theA
```

*dgDataControlOfIndex*
- get the dgDataControlOfIndex [ pIndex ]
- DATA GRID FORMS ONLY! Returns the long id of the data control associated with an index. If "cache controls" is not turned on then this property returns empty if the index has no control associated with it because it is offscreen.

- get the dgDataControlOfLine [ pLine ]
- Same as dgDataControlOfIndex but takes a line number as the parameter.

*dgDataOfIndex*
- get the dgDataOfIndex [ pIndex ]
- set the dgDataOfIndex [ pIndex ] of group "DataGrid" to pDataA
- Get or set the data associated with a particular index. The value is the array assigned to that index. Note that setting the data of an index will cause the data grid to refresh the row associated with the index if it is visible on screen.

```
put the dgDataOfIndex[1] of group "DataGrid" into theMessageA
put theMessageA["message"] -- puts "hi"
```

*dgDataOfLine*
- get the dgDataOfLine [ pLine ]
- set the dgDataOfLine [ pLine ] of group "DataGrid" to pDataA
- Get or set the data associated with a particular line. The value is the array assigned to that line. Note that setting the data of a line will cause the data grid to refresh the line if it is visible on screen.

```
put the dgDataOfLine[1] of group "DataGrid" into theMessageA
put theMessageA["message"] -- puts "hi"
```

*dgFocus*
- set the dgFocus of group "DataGrid" to true
- Set to true to focus on a data grid.

*dgFormattedHeight*
- get the dgFormattedHeight
- Returns the formatted height of the data in the data grid. The table header is not included in the formatted height.

*dgFormattedWidth*
- get the dgFormattedWidth
- Returns the formatted width of the columns in a data grid. This is only useful in tables as forms don't scroll horizonally.

*dgNumberOfLines*
- Returns the number of lines displayed in the data grid.

*dgNumberOfRecords*
- get the dgNumberOfRecords
- set the dgNumberOfRecords of group "DataGrid" to 20
- Getting the dgNumberOfRecords is the same as getting the dgNumberOfLines. Setting the dgNumberOfRecords has a special significance however. If you set the dgNumberOfRecords then you are telling the data grid that you know how many total records there are and you are going to supply the data for each record on an as-needed basis. This is useful when you have data in a database cursor that you would like to feed into the data grid.

After setting this property the data grid will send the GetDataForLine message to the data grid whenever it needs to display data for a particular line. You can define this command in the data grid script or elsewhere in the hierarchy. The definition is as follows:

```
command GetDataForLine pLine, @pDataA
```

end GetDataForLine

You should fill pDataA with the appropriate data based on the line of data being requested. pDataA should not have a numeric index. It is the array that would be assigned to one of the numeric indexes if you were assigning the dgData property.

*dgText*
- get the dgText [pIncludeColumnNames]
- set the dgText [pFirstLineContainsHeaders] of group "DataGrid" to pText
- The data grid works with arrays behind the scenes but in the interest of making life easier for some folks there is a dgText property. The dgText property always reflects the same value as the dgData but in tab delimited form.

*pText* is assumed to be a collection of data where each row is delimited by the return character and each item is delimited by a tab. You can map each item of each line in *pText* to a particular key in an array (and thus a table column) by passing in true for *pFirstLineContainsHeaders*. If true then the data grid will extract the first line of *pText* and use the values for the internal key/column names. The default value for *pFirstLineContainsHeaders* is false.

If you set the dgText of a **data grid table** then all data will be imported and assigned to the appropriate column depending on the value of *pFirstLineContainsHeaders.* Normally you should set this property to true and provide the header that maps each item of each line to a specific column. Note that if *pFirstLineContainsHeaders* is true then the columns must already exist in your data grid table in order to be displayed.

If *pFirstLineContainsHeaders* is false then the *columns* property of the data grid is used for mapping. For example, the first item of a line of *pText* would be assigned to the column that appears on the first line in the *columns* property of the data grid. If line 1 of *pText* contains more items than there are columns in the table then new columns are added. Any new columns are named "Col 1", "Col 2", etc.

If you set the dgText property of a **data grid form** then the data will be imported but it is up to you to modify your **Row Template Behavior** to display the imported data correctly. If *pFirstLineContainsHeaders* is false then each item of each line in *pText* will be named "Label X" (where X is the item number) in the array that is passed to FillInData.

When retrieving the dgText property you can include the column names in the first line by setting *pIncludeColumnNames* to true.

*dgHilitedIndexes*
*dgHilitedIndex*
- get the dgHilitedIndexes

- set the dgHilitedIndexes of group "DataGrid" to pIndex
- Returns a comma delimited list of the indexes that are currently selected.

*dgHilitedLines*
*dgHilitedLine*
- get the dgHilitedLines
- set the dgHilitedLines of group "DataGrid" to pLine
- Returns a comma delimited list of the line numbers that are currently selected.

*dgHScroll*
- get the dgHScroll
- set the dgHScroll to of group "DataGrid" pInteger
- Get/set the horizontal scroll of the data grid. This only applies to tables as forms do not scroll horizontally.

*dgHScrollPercent*
- get the dgHScrollPercent
- set the dgHScrollPercent of group "DataGrid" to pPercent
- Get/set the percentage of the horizontal scroll. A number between 0 and 1. This only applies to tables as forms do not scroll horizontally.

*dgIndexes*
- get the dgIndexes
- Returns the internal list of indexes in the order in which they appear in the data grid. Indexes are the numeric indices used when setting the dgData property.

*dgIndexOfLine*
- put the dgIndexOfLine [ pLine ]
- Returns the index associated with the given line.

*dgVScroll*
- get the dgVScroll
- set the dgVScroll of group "DataGrid" to pInteger
- Get/set the vertical scroll of the data grid.

*dgVScrollPercent*
- get the dgVScrollPercent
- set the dgVScrollPercent of group "DataGrid" to pPercent
- Get/set the percentage of the vertical scroll. A number between 0 and 1.

*dgVisibleLines*

- Returns the first and last line being displayed in the data grid as a comma delimited list. Useful if you want to provide visual feedback as to which lines are being displayed.

## Commands

The following are commands you can issue to the data grid. To issue a command you can use 'dispatch' (introduced in 3.5) or 'send'. I prefer 'dispatch' as it has the nice 'with' syntax for sending parameters.

*using dispatch*
put "value" into theArray["property"]
dispatch "AddData" to group "DataGrid" with theArray

*using send*
put "value" into theArray["property"]
send "AddData theArray" to group "DataGrid"

--------------------------

*AddData*
- AddData pDataArray, pLine
- Use this command to add data to the data grid after you have already populated it by setting the dgData or dgText.
pDataArray is an array of custom data to add to the data grid and pLine is the line number where it should be added. All data appearing at or after pLine will be shifted down 1. You will not overwrite any data. If pLine is empty then the data will be added to the end of the existing data.

If the data is successfully added then the index of the data will be returned in *the result*, otherwise an error is returned.

*AddLine*
- *AddLine* pText, pColumns, pLine
- Use this command to add tab delimited text to the data grid after you have already populated it by setting the dgData or dgText.
pText is tab delimited text to add to the data grid. pColumns is a cr delimited list of column names that text items map to. pLine is the line number where it should be added. All data appearing at or after pLine will be shifted down 1. You will not overwrite any data. If pColumns is empty then the "columns" property of the data grid is assumed. If pLine is empty then the data will be added to the end of the existing data.

*DeleteIndex*

*DeleteIndexes*
- DeleteIndexes pIndexes
- Deletes the specified indexes from the data grid. pIndexes is a comma delimited list of integers.

*DeleteLine*
*DeleteLines*
- DeleteLines pLines
- Deletes the specified lines from the data grid. pLines is a comma delimited list of integers.

*EditCell*
- EditCell pColumnName, pLineNo
- Sends the *EditValue* message to the column control for column pColumnName of line pLineNo. The default behaviors for columns handle the *EditValue* message and open the cell for editing by calling *EditFieldText*. Can only be used with tables.

*EditCellOfIndex*
- EditCellOfIndex pColumnName, pIndex
- Same as EditCell but uses an index rather than a line number to locate the line to edit.

*EditFieldText*
- EditFieldText pField, pIndex, pKey
- This command will dynamically create an editable field for editing the contents of pField (pass in the long id of a field for pField). Calling *EditFieldText* will trigger additional messages related to field editing.

Scenario 1: Pass in one parameter
If you just pass in pField and leave pIndex and pKey empty then the data grid behaves as follows:
1) Creates field editor
2) Assign text of pField to field editor.
3) Sends *preOpenFieldEditor pFieldEditor* to pField. pFieldEditor is the long id of the field editor created in step 1.

When editing stops (focus leaves field, user presses escape key, etc.) the message *DeleteFieldEditor* is sent to the data grid. This in turn sends *CloseFieldEditor pFieldEditor* to pField if the user changed the content or *ExitFieldEditor pFieldEditor* if no change was made. You can use these messages to save any changes the user made in *pFieldEditor*.

Scenario 2: Pass in all three parameters
If you pass in all three parameters (pField, pIndex, pKey) then the data grid will automatically save any changes made while editing. The new value will be assigned to the key pKey for index pIndex in the dgData array of the data grid.

In this scenarios *CloseFieldEditor pFieldEditor* and *ExitFieldEditor pFieldEditor* are still sent. The difference is that after *CloseFieldEditor* is sent to pField the contents of *pFieldEditor* are saved in the dgData. If for any reason you do not want the data to be saved then you can return "cancel" from *CloseFieldEditor*.

Note: If the user presses the tab key while editing and the autotab property of pField is true then the message *OpenNextFieldEditor pDirection* is sent to pField. If you don't handle this message and the data grid is a table then the data grid automatically opens the next cell for editing. For data grid forms you can handle this message in order to open another field for editing. You could call *EditKeyOfIndex* for example.

Note 2: The default behavior for pFieldEditor is located in button "Field Editor" of stack "revDataGridLibrary". If you want to override this behavior then you can assign the behavior of pFieldEditor to another button in the *preOpenFieldEditor* message.

*EditKey*
- EditKey pKey, pLineNo
- Sends the *EditValue pKey* message to the row control for line pLineNo. Handle the *EditValue* message in your data grid form row behavior in order to open a field in the row for editing. See *EditFieldText*.

*EditKeyOfIndex*
- EditKeyOfIndex pKey, pIndex
- Same as EditKey but uses an index rather than a line number to located the line to edit.

*FindIndex / FindLine*
- FindIndex pKey, pSearchString
- Search for data in pKey that matches pSearchString. pKey is one of the custom defined keys you defined for your data. pSearchingString is the value to look for in that key. You can pass multiple pKey=pSearchString combinations to match multiple criteria.

```
-- Find the index where "message" is "hi"
dispatch "FindIndex" to group "DataGrid" with "message", "hi"
put the result into theIndex
```

Note that pKey can also be an array if you want to use array-valued array indexing to locate pSearchingString.

*RefreshIndex*
- RefreshIndex pIndexes
- Redraws row associated with pIndexes using latest data. Use this command in conjunction with

SetDataOfIndex. pIndexes can be a comma delimited list of indexes.

*RefreshLine*
- RefreshLine pLines
- Redraws row using latest data. Use this command in conjunction with SetDataOfLine. pLines can be a comma delimited list of lines.

*ScrollIndexIntoView*
- ScrollIndexIntoView pIndex
- Scrolls the data grid so that the line associated with pIndex in the internal data array is in view.

*ScrollLineIntoView*
- ScrollLineIntoView pLine
- Scrolls the data grid so that pLine is in view.

*SelectAll*
- Selects all lines in the data grid.

*SetDataOfIndex*
- *S*etDataOfIndex pIndex, pKey, pValue
- Updates the key pIndex in the internal data array. If pKey is empty then pValue should be an array. pValue will be assigned to key pIndex in the internal data array. If pKey is not empty then pValue will be assigned to key pKey of key pIndex in the internal data array. The data grid display will not be updated to reflect the new values. To update the display call RefreshIndex. Use dgDataOfIndex if you want to automatically refresh the data grid when you update the data.

*SetDataOfLine*
- *S*etDataOfLine pLine, pKey, pValue
- Updates the internal array of line pLine in the data grid. See notes for SetDataOfIndex.

*SortByColumn*
- SortByColumn pColumn
- Pass in a column to sort by. The current sort properties of the column will be used to perform the sort.

*SortDataByKey*
- SortDataByKey pArrayKey, pSortType, pDirection, pCaseSensitive

SortDataByKey is the underlying command that all column sorts call. It can be used to sort data grid forms. pArrayKey is one the keys you created when you assigned set dgData property. If you used the dgText property then the key will be "Label 1" or "Label 2", etc. pSortType, pDirection and pCaseSensitive all reflect the equivalent parameters for the built-in 'sort container' command in Revolution. pSortType is "text", "numeric', "dateTime" or "international". pDirection is "ascending" or "descending". pCaseSensitive is true/false.

*RefreshList*
- Redraws the data displayed in the data grid.

*ResetControl*
- Resets the control by clearing out any data.

*ResetList*
- Redraws the data grid data after having copied in fresh copies of any templates.

*ResizeToFit*
- Used internally when the rect of the data grid changes. Normally you will not need to call this handler as setting the rect of the data grid will trigger it.

## Functions

*ColumnControlOfIndex*
- ColumnControlOfIndex (pColumnName, pIndex)
DATA GRID TABLES ONLY! Returns the control for the column of index pIndex in the Data Grid. If "cache" controls" is not true then this may return empty if the row the index is displayed in is currently offscreen.

*GetDataOfIndex*
- GetDataOfIndex (pIndex, pKey)
- Retrieves the internal array for key pIndex of the internal array in the data grid. If pKey is empty then the array associated with key pIndex in the internal data array will be returned. If pKey is not empty then the value of that key for key pIndex of the internal data array will be returned.

*GetDataOfLine*
- GetDataOfLine (pLine, pKey)
- Retrieves the internal array associated with line pLine in the data grid. See notes for GetDataOfIndex.

## Messages Sent

*selectionChanged*
- selectionChanged pHilitedIndexes, pPreviouslyHilitedIndexes
- Sent whenever the user changes the selection through some sort of user interaction. Handle this message in order to update your UI when the user makes a selection.

```
-- example that would be placed in the data grid script.
on selectionChanged pHilitedIndex, pPrevHilitedIndex
```

```
    put the dgDataOfIndex [ pHilitedIndex ] of me into theSelDataA
    uiViewRecordOfID theSelDataA["id"]
end selectionChanged
```

*EditValue*

- EditValue pKey
- The EditValue message is sent to a table column control when EditCell or EditCellOfIndex is called. In this scenario there is no pKey parameter since the key is the column associated with the column control the message was sent to. Tip: use the dgColumn property of a column template to get the name of a column in the column control.

The EditValue message is sent to a row when EditKey or EditKeyOfIndex is called. In this case the first parameter is the name of the key in dgData that should be edited. Normally you will call EditFieldText from within the EditValue message in order to open a field editor for changing the contents.

## Template Custom Properties & Messages

When you are coding behaviors for custom templates the following properties and messages are applicable.

### The Following Messages Are Sent To Your Custom Templates

*FillInData pData*

- The FillInData message is where you move data for a row or column into the controls for that row or column. Normally you will just assign data to controls in the template. You most likely will not resize any of the controls at this point.

If your data grid is of type "form" then pData will be an array holding the row values. If your data grid is of type "table" then pData is the value for the column that is being populated.

Data Grid row example:

```
on FillInData pDataA
   ## Assign FirstName value to the "FirstName" field of this row template
   set the text of field "FirstName" of me to pDataA["FirstName"]
end FillInData
```

Data Grid table example:

```
on FillInData pData
   ## Assign Column value to field 1 of this column template
   set the text of field 1 of me to pData
end FillInData
```

*LayoutControl pControlRect*

- LayoutControl is sent to your custom template when it is time to position all of the controls. pControlRect is the rectangle that the data grid has resized your control to. This is useful for knowing the left (item 1 of pControlRect), top (item 2 of pControlRect), right (item 3 of pControlRect) and bottom (item 4 of pControlRect) bounds you can position controls at. Note that if you have a data grid form that does not have fixed height set to true then you can ignore the botom (item 4 of pControlRect) and make your control as tall as you need.

Example:

```
on LayoutControl pControlRect
   set the rect of field "FirstName" to pControlRect
```

**end** LayoutControl

## The following custom properties are available to your template behavior scripts by default

*dgLine*
- The line (or row) number that is being displayed in the copy of the template.

*dgIndex*
- The index used to uniquely identify the record being displayed in the copy of the template.

*dgColumn*
- When working with a data grid table you can use *the dgColumn of me* in a custom template behavior to get the name of the column the instance of the template is associated with. To use this property in other scripts you use *the target* or *the mousecontrol* as the target as well.

*dgColumnNumber*
- When working with a data grid table you can use *the dgColumnNumber of me* in a custom template behavior to get the column number relative to all visible columns. To use this property in other scripts you use *the target* or *the mousecontrol* as the target as well.

## The following custom properties are defined in the behavior script for your template

dgDataControl (getProp)
- Your template should 'return the long id of me' in this getprop handler. This helps the data grid identify your row template. THIS IS REQUIRED FOR YOUR DATA GRID TEMPLATE TO WORK PROPERLY!!

dgHilite pBoolean (setProp)
- This property is set by the data grid to set the highlighted state of the row or column control. If pBoolean is "true" then the control should be highlighted. If "false" then it should not be.