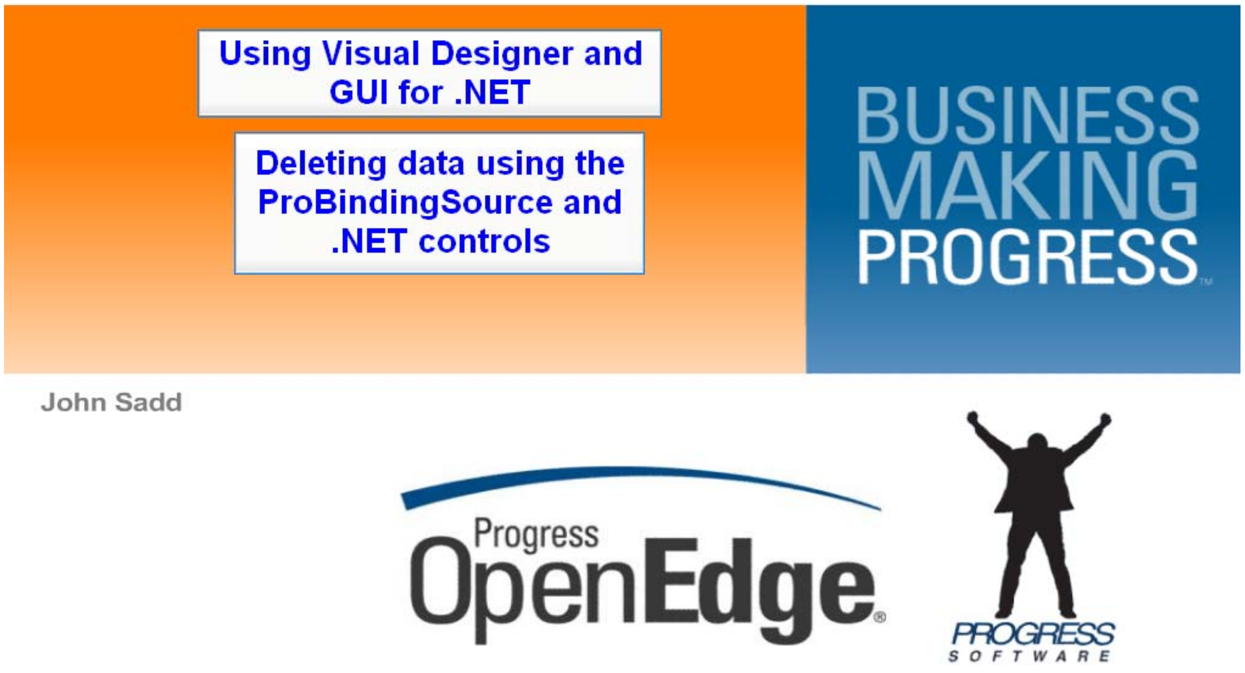
DELETING DATA WITH THE PROBINDINGSOURCE AND .NET CONTROLS

John Sadd
Fellow and OpenEdge Evangelist
Document Version 1.0
March 2010



Using Visual Designer and GUI for .NET

Deleting data using the ProBindingSource and .NET controls

BUSINESS MAKING PROGRESS™

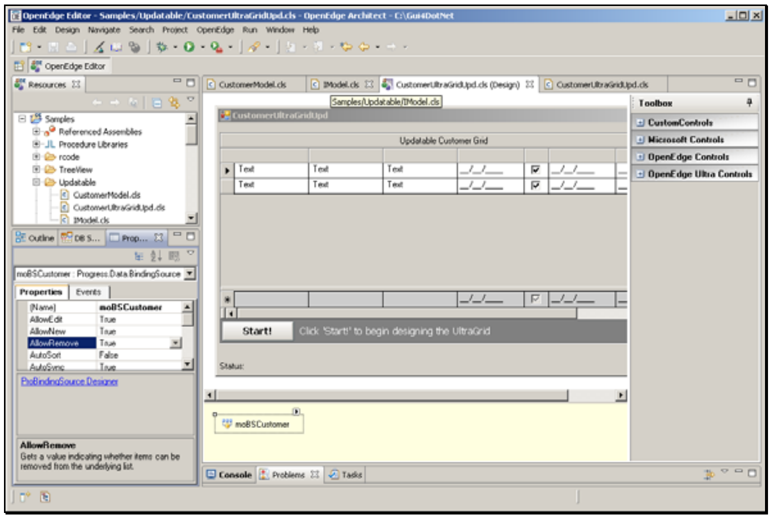
John Sadd

Progress OpenEdge®

PROGRESS SOFTWARE

DISCLAIMER

This document accompanies another in a series of presentations on updating data using the ProBindingSource and .NET controls. It introduces the binding source and UltraGrid properties and events you use to manage row deletes. First let's take another look at the ProBindingSource properties. As with **AllowEdit** and **AllowNew**, there's a property called **AllowRemove** that tells the binding source whether to allow row deletes through the query or DataSet it's bound to.



moBSCustomer : Progress.Data.BindingSource

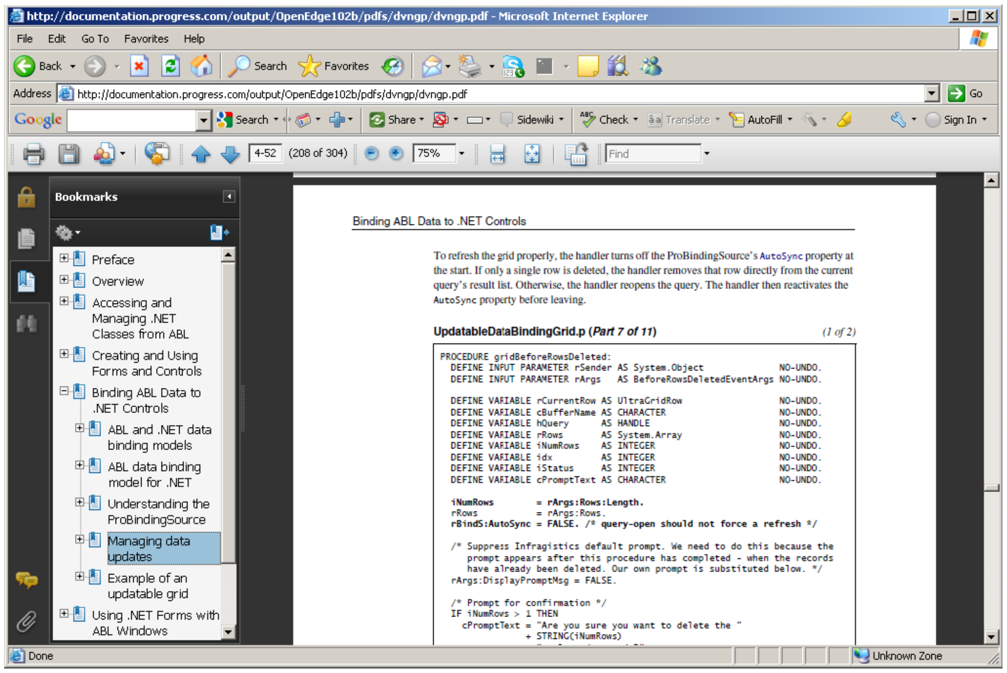
(Name)	**moBSCustomer**
AllowEdit	True
AllowNew	True
AllowRemove	True
AutoSort	False
AutoSync	True

AllowRemove
Gets a value indicating whether items can be removed from the underlying list.

And like the others, it's **True** by default, so it can be left that way to allow row deletes through the binding source. Remember that you can set it to **False** for any form where data deletes should not be allowed, and it will manage the UI controls' behavior for you. Once again, I'll start with the interface for my Model classes, to add a method definition to manage deletes in the temp-table and pass them on to the database table itself. The new method is called **DeleteModelRow**, and like the others defined in earlier sessions in this series, it takes the buffer name as a parameter in case the Model holds a ProDataSet with more than one table.

```
INTERFACE Updatable.IModel:
  …
  METHOD PUBLIC LOGICAL CreateModelRow (INPUT pcBufferName AS CHARACTER ).
  METHOD PUBLIC LOGICAL CancelCreateModelRow (INPUT pcBufferName AS CHARACTER ).
  METHOD PUBLIC LOGICAL DeleteModelRow (INPUT pcBufferName AS CHARACTER ).
END INTERFACE.
```

I add the method to the **CustomerModel** example class, using the support that Architect provides for **Override / Implement Methods** under the **Source** menu to generate the right code skeleton for me. It's possible to code support for deleting multiple rows at a time. The user can select more than one row and then press **Delete**. In this case the code in the View would have to pass key information from the View down to the Model to tell it how to identify the rows in the table that correspond to the rows in the grid that the user selected for deletion. There is an example of that kind of code in the chapter on data binding in the ***OpenEdge Development: GUI for .NET Programming*** book, as shown here:


http://documentation.progress.com/output/OpenEdge102b/pdfs/dvngp/dvngp.pdf - Microsoft Internet Explorer
File Edit Go To Favorites Help
Back Search Favorites
Address http://documentation.progress.com/output/OpenEdge102b/pdfs/dvngp/dvngp.pdf Go
Google Search Share Sidewiki Check Translate AutoFill Sign In
4-52 (208 of 304) 75% Find
Bookmarks
Preface
Overview
Accessing and Managing .NET Classes from ABL
Creating and Using Forms and Controls
Binding ABL Data to .NET Controls
ABL and .NET data binding models
ABL data binding model for .NET
Understanding the ProBindingSource
Managing data updates
Example of an updatable grid
Using .NET Forms with ABL Windows
Done
Unknown Zone

Binding ABL Data to .NET Controls

To refresh the grid properly, the handler turns off the ProBindingSource's AutoSync property at the start. If only a single row is deleted, the handler removes that row directly from the current query's result list. Otherwise, the handler reopens the query. The handler then reactivates the AutoSync property before leaving.

UpdatableDataBindingGrid.p (*Part 7 of 11*) *(1 of 2)*

```
PROCEDURE gridBeforeRowsDeleted:
  DEFINE INPUT PARAMETER rSender AS System.Object               NO-UNDO.
  DEFINE INPUT PARAMETER rArgs   AS BeforeRowsDeletedEventArgs NO-UNDO.

  DEFINE VARIABLE rCurrentRow AS UltraGridRow                   NO-UNDO.
  DEFINE VARIABLE cBufferName AS CHARACTER                      NO-UNDO.
  DEFINE VARIABLE hQuery      AS HANDLE                         NO-UNDO.
  DEFINE VARIABLE rRows       AS System.Array                   NO-UNDO.
  DEFINE VARIABLE iNumRows    AS INTEGER                        NO-UNDO.
  DEFINE VARIABLE idx         AS INTEGER                        NO-UNDO.
  DEFINE VARIABLE iStatus     AS INTEGER                        NO-UNDO.
  DEFINE VARIABLE cPromptText AS CHARACTER                      NO-UNDO.

  iNumRows        = rArgs:Rows:Length.
  rRows           = rArgs:Rows.
  rBindS:AutoSync = FALSE. /* query-open should not force a refresh */

  /* Suppress Infragistics default prompt. We need to do this because the
     prompt appears after this procedure has completed - when the records
     have already been deleted. Our own prompt is substituted below. */
  rArgs:DisplayPromptMsg = FALSE.

  /* Prompt for confirmation */
  IF iNumRows > 1 THEN
    cPromptText = "Are you sure you want to delete the "
                + STRING(iNumRows)
```

But my goal here is just to show you how the basic properties work, so I limit the support to deleting one selected row at a time. This simplifies things, because if there's just one selected row it will be positioned to in the temp-table query automatically.

```
METHOD PUBLIC LOGICAL DeleteModelRow( INPUT pcBufferName AS CHARACTER ):

        DELETE ttCustomer.
        httCustQuery:DELETE-RESULT-LIST-ENTRY ().
        IF SaveData("ttCustomer") THEN
            RETURN TRUE.
        ELSE RETURN FALSE.

END METHOD.
```

The method in the Model just needs to delete the current row, and use the ABL **`DELETE-RESULT-LIST-ENTRY`** method to remove it from the query without reopening the query. Here the ProBindingSource **AutoSync** property that was introduced in the presentation on row creates causes the grid to reflect the row deletion automatically. After the execution of these two lines of ABL, the row is gone from the grid, from the temp-table, and from the query, but the method has to invoke the existing **SaveData** method to apply the delete to the data source – the database table – itself. Looking at the code already in **SaveData** allows us to see if there's a change that has to be made to what it does.

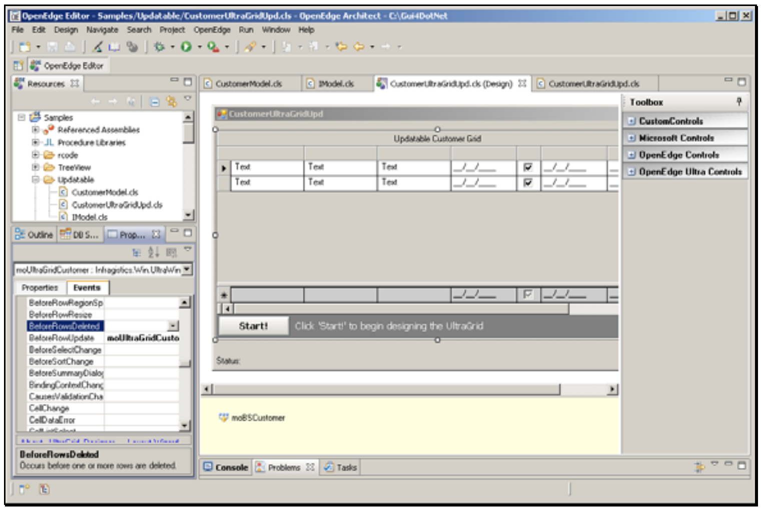
In a ProDataSet, when I delete a row, it's removed from the temp-table itself, which is what **`hCustBuffer`** points to in the code shown below, but the record of the delete is kept in the ProDataSet's Before Table, which is what **`hBeforeBuffer`** points to, so that **`SAVE-ROW-CHANGES`** knows what row to delete from the database table. Since the Before Table and the **`SAVE-ROW-CHANGES`** method apply to all forms of data management, including deletes as well as updates and creates, these handles and the table they point to apply to the row delete case as well as others.

```
METHOD PUBLIC LOGICAL SaveData( INPUT pcBufferName AS CHARACTER ):

    DEFINE VARIABLE hCustBuffer   AS HANDLE NO-UNDO.
    DEFINE VARIABLE hBeforeBuffer AS HANDLE NO-UNDO.

        …

        hCustBuffer = BUFFER ttCustomer:HANDLE.
        hBeforeBuffer = hCustBuffer:BEFORE-BUFFER.
```

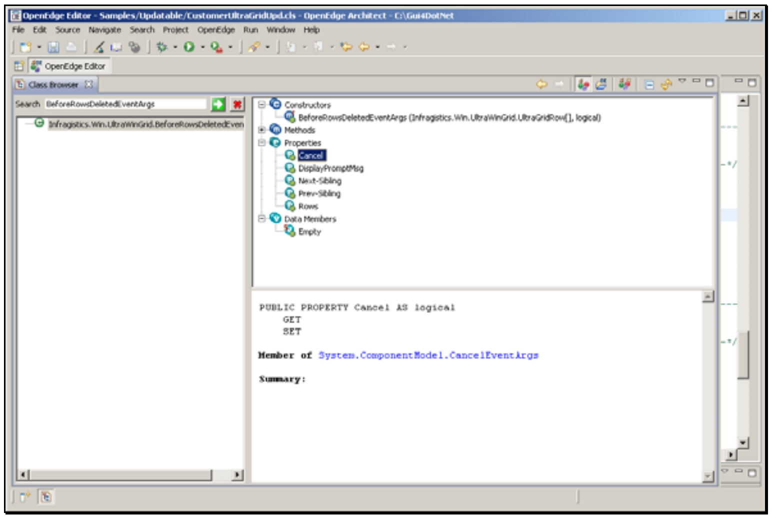
However, the sample validation code that follows doesn't apply if I'm deleting the whole row, and in fact it will fail, because the row is no longer in **ttCustomer**. So I have to add a check to skip any validation if the change being made is a delete. That's recorded in the DataSet's Before Table as a **ROW-STATE** of **ROW-DELETED**. So the code has to be changed to skip the validation in the case of a delete.

```
    IF hBeforeBuffer:ROW-STATE NE ROW-DELETED THEN
    DO:
        IF ttCustomer.CustomerBirthCountry NE "USA" AND
           ttCustomer.CustomerBirthCountry NE "Germany" THEN
        DO:
           MESSAGE "Invalid Birth Country value, must be USA or Germany."
               VIEW-AS ALERT-BOX.
           hBeforeBuffer:REJECT-ROW-CHANGES ().
           RETURN FALSE.
        END.
     END.
```

For a single row delete that's all that's needed in the Model. Going back to the form and the grid, there's another useful **UltraGrid** event called **BeforeRowsDeleted**, which is what I need to subscribe to in order to add support for deletes to the user interface.



In the generated skeleton code for the event handler, you can see that the **EventArgs** parameter passed in is a subclass called **BeforeRowsDeletedEventArgs**. Looking at that in the Class Browser, you can see first of all that, as in the **EventArgs** class for row creates, there's a **Cancel** property that needs to be set to tell the handler whether the delete completed successfully or needs to be backed out.


OpenEdge Editor - Samples/Updatable/CustomerUltraGridUpd.cls - OpenEdge Architect - C:\Gui4DotNet
File Edit Source Navigate Search Project OpenEdge Run Window Help
OpenEdge Editor
Class Browser
Search BeforeRowsDeletedEventArgs
Infragistics.Win.UltraWinGrid.BeforeRowsDeletedEven
Constructors
BeforeRowsDeletedEventArgs (Infragistics.Win.UltraWinGrid.UltraGridRow[], logical)
Methods
Properties
Cancel
DisplayPromptMsg
Next-Sibling
Prev-Sibling
Rows
Data Members
Empty

```
PUBLIC PROPERTY Cancel AS logical
    GET
    SET

Member of System.ComponentModel.CancelEventArgs

Summary:
```

And specific to deletions, there's a **DisplayPromptMsg** property, a **Logical** value that determines whether you get a default message asking you to confirm the delete. The grid will display one for you if you set this property to **True**, but it turns out that the message doesn't appear at the right time to allow you to cancel the row delete from the temp-table, so you should normally set the property to **False**. The method implementation below shows a simple example of where to put your own message. In addition, there's a **Rows** property that holds a collection of, in this case, all the rows being deleted. The event is called **BeforeRowsDeleted**, so it is in fact prepared to allow more than one delete at a time, though that's not shown in this example.

Here is the code for the **BeforeRowsDeleted** event handler that uses these **EventArgs** properties:

```
METHOD PRIVATE VOID moUltraGridCustomer_BeforeRowsDeleted(
    INPUT sender AS System.Object,
    INPUT e AS Infragistics.Win.UltraWinGrid.BeforeRowsDeletedEventArgs ):

    DEFINE VARIABLE oRow            AS UltraGridRow NO-UNDO.
    DEFINE VARIABLE cBufferName     AS CHARACTER    NO-UNDO.

    e:DisplayPromptMsg = FALSE. /* Don't use grid's default are-you-sure msg */
    IF e:Rows:Length > 1 THEN
    DO:
      MESSAGE "Deleting multiple rows at once is not supported." VIEW-AS ALERT-BOX.
      e:Cancel = TRUE.
    END.
    ELSE DO:
        MESSAGE "Are you sure you want to delete the selected row?"
            VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO UPDATE lConfirmDelete
                AS LOGICAL.
        IF NOT lConfirmDelete THEN
            e:Cancel = TRUE.
        ELSE DO:
            oRow = CAST(e:Rows:GetValue(0), UltraGridRow). /* One row deleted. */
            cBufferName = oRow:Band:Key.
            IF moCustomerModel:DeleteModelRow (cBufferName) THEN
                e:Cancel = FALSE.
            ELSE e:Cancel = TRUE.
        END.
    END.
    RETURN.

END METHOD.
```

First the code turns off the default "Are you sure?" message by setting **DisplayPromptMsg** to **False**, so that the method can control what the message says and when it appears. Then it examines the **Rows** property, which is a .NET collection. One of the standard properties of a collection is **Length**, the number of elements in the collection, so the code checks the **Length** to make sure that only one row is marked for deletion. If not, it sets the **Cancel** property to **True** to signal that the delete didn't succeed. Then it supplies a custom delete confirmation message, and sets the **Cancel** flag accordingly.

One of the standard collection methods is **GetValue**, which takes a zero-based index into the collection. Here the statement…

```
oRow = CAST(e:Rows:GetValue(0), UltraGridRow).
```

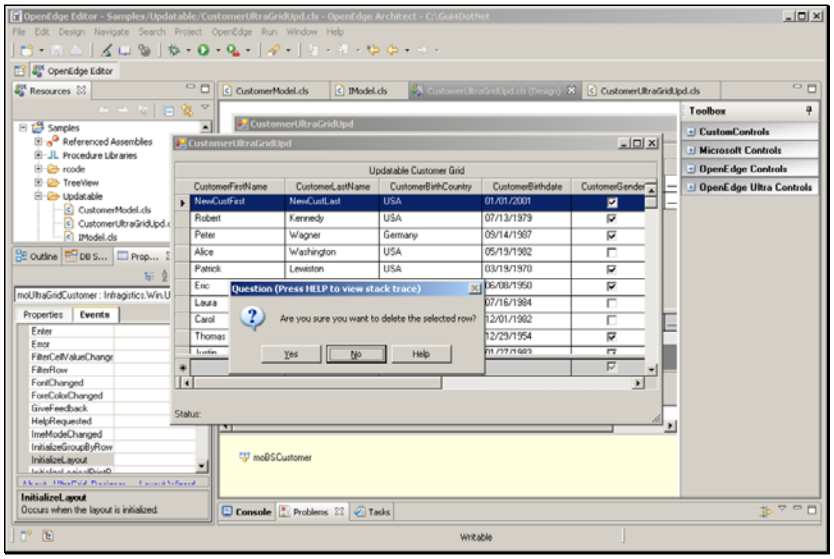
…returns the first (and in this case, only) row in the **Rows** collection. The ABL **CAST** function tells the compiler to treate the object reference returned by **GetValue** as an instance of an **UltraGridRow**, which is what the **Rows** collection contains. If the method were supporting the deletion of multiple rows at a time, it would have to loop through the rows in the collection (since the collection contains the rows the user has selected for deletion), extract key values from each row, and pass those to a delete method in the Model so that it could identify which rows had been selected. The present example only accesses the one selected row to extract the key value from the Band it's in, which holds the buffer name for the data displayed in that band:

```
cBufferName = oRow:Band:Key.
```

The buffer name would be important if the binding source and the grid supported a DataSet with more than one table. Finally, the method invokes the delete method in the Model class:

```
IF moCustomerModel:DeleteModelRow (cBufferName) THEN
    e:Cancel = FALSE.
ELSE e:Cancel = TRUE.
```

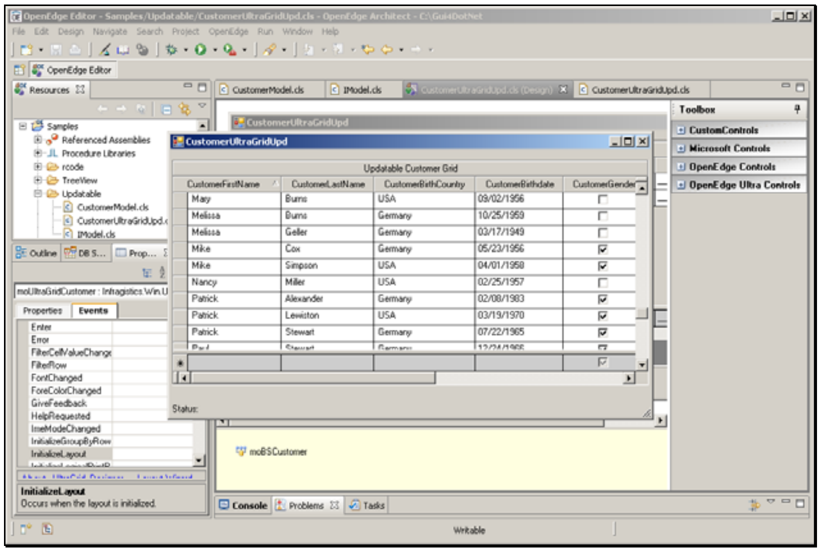
After saving and re-running the form class, I can select, for instance, the test row that I added for my support of row creates in another presentation. The grid recognizes the keyboard **Delete** key as a delete request. Here's the **DeleteModelRow** method's confirmation message:



CustomerFirstName	CustomerLastName	CustomerBirthCountry	CustomerBirthdate	CustomerGender
NewCustFirst	NewCustLast	USA	01/01/2001	✓
Robert	Kennedy	USA	07/13/1979	✓
Peter	Wagner	Germany	09/14/1987	✓
Alice	Washington	USA	05/19/1982	☐
Patrick	Lewiston	USA	03/19/1970	✓
Eric			06/08/1950	✓
Laura			07/16/1984	☐
Carol			12/01/1982	☐
Thomas			12/29/1954	✓

Are you sure you want to delete the selected row?

After clicking Yes, and just to make sure the row is gone, I can sort by **CustomerFirstName**, and scroll down and look for a customer named **NewCustFirst**, and confirm that it's not there:



CustomerUltraGridUpd

Updatable Customer Grid

CustomerFirstName	CustomerLastName	CustomerBirthCountry	CustomerBirthdate	CustomerGender
Mary	Burns	USA	09/02/1956	☐
Melissa	Burns	Germany	10/25/1959	☐
Melissa	Geller	Germany	03/17/1949	☐
Mike	Cox	Germany	05/23/1956	✓
Mike	Simpson	USA	04/01/1958	✓
Nancy	Miller	USA	02/25/1957	☐
Patrick	Alexander	Germany	02/08/1983	✓
Patrick	Lewiston	USA	03/19/1970	✓
Patrick	Stewart	Germany	07/22/1965	✓

Status:

To summarize, in this session I showed you the ProBindingSource **AllowRemove** property, which needs to be **True** for the binding source to allow deletes through the query it's connected to. If you allow just a single row delete at a time, you just delete that row in the underlying table -- because the binding source positions to it when it's selected in the grid -- and then delete the corresponding result list entry from the table's query. If you want to support deleting multiple rows, walk through the **Rows** collection and pass key information that identifies each row to the code in the Model that can use that to delete them from its temp-table and the underlying data source.