UPDATING DATA WITH .NET CONTROLS AND A PROBINDINGSOURCE

John Sadd Fellow and OpenEdge Evangelist Document Version 1.0 March 2010





DISCLAIMER

Certain portions of this document contain information about Progress Software Corporation's plans for future product development and overall business strategies. Such information is proprietary and confidential to Progress Software Corporation and may be used by you solely in accordance with the terms and conditions specified in the PSDN Online (http://www.psdn.com) Terms of Use (http://psdn.progress.com/terms/index.ssp). Progress Software Corporation reserves the right, in its sole discretion, to modify or abandon without notice any of the plans described herein pertaining to future development and/or business development strategies. Any reference to third party software and/or features is intended for illustration purposes only. Progress Software Corporation does not endorse or sponsor such third parties or software.

This paper accompanies a two-part presentation that extends other materials on data binding and using the ProBindingSource control, by showing you simple ways to handle update operations. Here I introduce more of the remaining **ProBindingSource** properties, methods and events that support updates, and a few of the basic UI control events that you can interact with as well. As you can see here in the **Resources** View, I've created a new subdirectory in my project called **Updatable** where I'll put my new code samples:

📴 OpenEdge Editor - OpenEdge Architect - C:\Gui4Dothet	
File Edit Navigate Search Project OpenEdge Run Window Help	
12 • 12 △ ∡山 20 歩 • 0 • 24 • タ • 2 • 21 • 1 + + + + + +	
🗈 🖧 OpenEdge Editor	
🗱 Resources 🖂 🧧 🗖	
🗏 🛱 Samples 🔺	
B 30 Referenced Assemblies	
B-JL Procedure Libraries	
8 😓 rcode	
8 😓 Tretilew	
B 😂 Understele	
- C. propath	
- appModel.74bi	
- B assembles.xml	
Property Value	
derive: false	
edtabl true	
last mc January 14, 2010 Inled false	
location (-)(QuHOotNet)Se	
name Updatable	
path /Samples/Updatable	
🕒 Console 🔝 Problems 22 🖉 Tasks	* □
T° E Samples/Lipidatable	

That directory corresponds to the notion of a package, in object-oriented terms, for organizing the code in my project. To show you how to specify the package when you create new source files, I start by creating a new ABL Interface. Because I had selected the **Updatable** folder in the **Resources View**, that package name is already filled in for me as the default. As I've done before, I can just enter the name of the interface. This is a variation on the **IModel** interface that I've used before, so I can use the same name, because it will be stored in a new folder with all the rest of my sample code for trying out update operations:

🝞 OpenEdge Editor - OpenEdge Architect			_101×
File Edit Navigate Search Project Open			
] 🗈 • 🖩 🛆] 🔏 💷 🕲] 🏇 • •	Q • Q ₄ • [A ² •] ≥ × 8 × 6 + × 6 ×		
📑 💱 OpenEdge Editor	📧 New ABL Interface	XOX	
🗱 Resources 🕱 📃 🗖	Create a user-defined interface		- 0
	Optionally enter a description for the interface. This text will appear in the file header.	I	
🗄 🎒 Samples 📃			
	Package root: \Samples	Browse	
🗈 🗁 rcode	Package: Updatable	Browse	
Comparison TreeView Dedatable			
- dbconnection	Interface name: IModel		
A project	Description:	A	
- propath			
appModel.t4bl			
Assembliesml			
		*	
80 00 DP 20 0	Purpose:	-	
E 🕸 🕾 🎽			
Property Value			
Info derive: false			
editabil true		x	
last mc January 14, 2010			
linked false			
locatio C:\Gui4DotNet\Sa name Updatable			
path /Samples/Updatable	() Ensh	Cancel	
	Devi		
	onsole 🚺 Problems 🕴 🅢 Tasks		
📄 🎦 Samples/Updatable			

I start with the Interface code skeleton that Architect generates, paste in the three method definitions used in the read-only operations created in other sessions on the **ProBindingSource**, and then add a new method to support saving data, which takes a buffer name as a parameter, and returns a **LOGICAL** value to tell me whether the save succeeded or not.

USING Progress.Lang.*.
INTERFACE Updatable.IModel:
METHOD PUBLIC VOID FetchData (INPUT pcFilter AS CHARACTER).
METHOD PUBLIC VOID SortData (INPUT pcSort AS CHARACTER).
METHOD PUBLIC HANDLE GetQuery().
METHOD PUBLIC LOGICAL SaveData(INPUT pcBufferName AS CHARACTER).
END INTERFACE.

Next I create a new class based on the **CustomerModel** class, and place it into my **Updatable** package. The first change I need to make is to qualify the name of the interface it implements so that the compiler finds the right one.

CLASS Updatable.CustomerModel IMPLEMENTS Updatable.IModel:

Then I need to make a change to the **FetchData** method that tells the model class to populate its ProDataSet. A ProDataSet can keep track of changes you make after it's been filled with data from its data source. But before I fill it I have to make sure that the **TRACKING-CHANGES** property on the **ttCustomer** temp-table is set off. Then after the **FILL** is done, I can set the property to **True** so that any changes made from that point on will be tracked in the **ttCustomer** temp-table's before-table:

```
METHOD PUBLIC VOID FetchData( INPUT pcFilter AS CHARACTER ):
DEFINE VARIABLE cPrepare AS CHARACTER NO-UNDO.
TEMP-TABLE ttCustomer:TRACKING-CHANGES = FALSE.
cPrepare = "FOR EACH AutoEdge.Customer".
IF pcFilter NE "" THEN
cPrepare = cPrepare + " WHERE " + pcFilter.
QUERY qCustomer:QUERY-PREPARE (cPrepare).
DATASET dsCustomer:FILL().
httCustQuery:QUERY-OPEN ().
TEMP-TABLE ttCustomer:TRACKING-CHANGES = TRUE.
END METHOD.
```

Now I'm ready to accept updates from the user interface. Remember that the **Source** menu in Architect helps you make all sorts of edits to a source file, including new methods, constructors, and event and property definitions for a class. I need to add a new method, but because it's a method defined in my interface **IModel**, I can select the **Override / Implement Members** option to implement the **SaveData** method that I defined in the **Updatable** version of the Interface. I have Architect add it to the source file in alphabetical order, and tell Architect to generate an empty comments block at the top of the method.

00 OpenEdge Editor - Samples/Upda	table/Customer	ModeLcls - OpenEdge Architect - C:\Gui4DotNet	_IO ×
File Edit Source Navigate Search	Project OpenEdg	e Run Window Help	
] 🗈 • 🔛 🗅] 🔏 💷 🦦] 🕸	> • • • •	0E Override/Implement Members	
🔛 🎳 OpenEdge Editor		Generate stubs to override or implement members	
🥰 Resources 😫 📃 🗆	IModel.ds	Select this option to include a comment block with each generated	
	298 CC	member.	-
E 14 Samples	30	Select members to override or inclement:	
B-SP Referenced Assemblies	31 32	Customer: HANDLE) .	
-JL Procedure Libraries	32	Select All	
🖲 🗁 OOSamples 📃	34	E Deselect Al	
Code Prove	35 EN	SaveData(INPUT CHARACTER):LOGICAL	
E- 🔁 Updatable	3.6		
CustomerModel.cls	37 380 ME		
CustomerUltraGrid.	39		1
IModel.ds	40 DE		
doconnection	41		
	42		
	44		
<u> </u>	45		
Property Value	4.6	Insertion position : Alphabetical order	
derived false	47	Generate comments	_
editable true	48 49 EN		
last modifie January 29, 2010	50	Select which style of member code you would like to create:	
linked false location C:\Gui4DotNet\Sa	S18 ME	Generate exceptions for required members	
name CustomerModel.cl	52	Generate default values for required members	
path /Samples/Updatab	53 /* 54	he DataSet. */	
size 6,999 bytes	55 EX		
	56		-
	•	(?) Generate Cancel)
4 P	🕒 Console 🖾		9 - 🗆 🗖
10. G		Writable Insert 57:5	

To the default code I start out with, as generated by Architect, I first add a couple of variable definitions to point to the temp-table's buffer, and its before-buffer, where changes are kept track of:

```
METHOD PUBLIC LOGICAL SaveData( INPUT pcBufferName AS CHARACTER ):
DEFINE VARIABLE hCustBuffer AS HANDLE NO-UNDO.
DEFINE VARIABLE hBeforeBuffer AS HANDLE NO-UNDO.
```

Also, I put in a sanity check to make sure that the buffer name passed in is the right one. In this case there's only one temp-table in the DataSet, but in other cases there could be more than one, which is why the buffer name parameter is here:



And then I initialize the two buffer handle variables:

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

I want to have some simple logic in the Model to give me a way to show what happens if the user enters invalid data. My table has **CustomerBirthCountry** values of only the USA and Germany, so my check says that if any other value is entered, I reject the change to the temp-table, and return false to signal the error to the View:

```
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.
```

Note that the Model class expects the changes the user has made to have been saved to the DataSet's temp-table already. This is important because you don't want even simple logic like this in the View, which just handles the user interface. And you don't want the Model, which is in charge of the data, to know how to look up into the UI and see the values in the user interface controls. So the UI has to get the values assigned to the temp-table before it runs **SaveData**, and to do this, it uses the ProBindingSource as an intermediary. If there's no error the Model uses the ProDataSet's **Save-Row-CHANGES** method to save data back to the DataSource, the database table it came from. **Save-Row-CHANGES** itself could fail, because it does conflict checks if more than one user is changing the same row at the same time, and there might be underlying database triggers that could fail as well. So if **Save-Row-CHANGES** succeeds, the **ACCEPT-ROW-CHANGES** method keeps the changes in the temp-table and marks the change as complete, and I tell the UI that the save succeeded:

```
/* If we get here all client-side validation succeeded. */
    IF hBeforeBuffer:SAVE-ROW-CHANGES () THEN
    DO:
        hBeforeBuffer:ACCEPT-ROW-CHANGES ().
        RETURN TRUE.
    END.
```

But if **SAVE-ROW-CHANGES** returns an error, then **REJECT-ROW-CHANGES** scrubs the change from the temptable, and the method returns an error flag.

```
ELSE DO:

hBeforeBuffer:REJECT-ROW-CHANGES ().

RETURN FALSE.

END.

END METHOD.
```

That's a start to the changes to the Model class to handle the data management side of the update. Now let's look at the Form where we'll also make changes, which once again is based on a form used in the earlier data binding examples. If I select its **ProBindingSource** control, I can show you a few more of its properties. The **AllowEdit** property, for instance, determines whether updates through the binding source are enabled or not. You can also enable individual controls in the user interface, but this property lets you control updates programmatically from the binding source. It's true by default, so updates are enabled. but here I set it to false to see how it affects the user interface.

😿 OpenEdgeEditor - Samples/Updatable/ File Edit Design Navigate Search Project				ect - C:\Gui4DotN	iet				1	-1012
¹ • □ △ <u>∠</u> □ ⊗ ≫ • (•						
📑 🞳 OpenEdge Editor										
Resources 😫 🔍 🗆	Model	ds Custom	erModel.cls	SustomerUitz	aGridUpd.ds (Desig	n) 🛙				00
	7							10	oolbox	9
E 🖉 Samples	🖌 🔛 Cus	tomerUltraGridUj	pd						EustomEontrols	
B Seferenced Assemblies	1 _									-
Image: Procedure Libraries				Updatable Cu	astomer Grid				Microsoft Controls	_
e code									OpenEdge Controls	
E Contraction	▶ Te		ext	Test					OpenEdge Ultra Cor	trols
Updatable CustomerModel.cls	Te	nst Te	ext	Test		R				
te 01 CS 1 meSSCutome: Proyets Data Endergiouces Propertite Events Events Propertite Plane() Adorfferore Adorfferore Tota Adorfferore Tota Adorfferore Tota Adorffero Adorffero Adorffer		Start! Click BSCustomer	: 'Start!' to be	gin designing the	e UtraGrid		,	Ĩ		
Gets a value indicating whether items in the.	Consol	e 🖹 Problems 🕅	🙆 Tasks						* ₽	- 0
							J			

I save the form with that property setting, and run the form to see what happens to my grid. If I select a cell and try to type into it, nothing happens. So the **ProBindingSource** has effectively disabled any controls that are bound to it, which can be a more effective way to manage updates than enabling and disabling individual controls. But it's clear that I want to keep **AllowEdit** set to **True**, so I reset the property to its default.

Another property of the binding source that looks interesting is **AutoUpdate**. This does what the name implies: it will automatically do the binding source **Assign** for you, which in my case would write changes back to the temp-table. But it does not do any error checking or validation or any of the work to get changes back to the actual data source, so it's not recommended except for quick prototyping. That's why it's **False** by default, and I leave it that way.

Next let me look at the events on the grid to see what I want to intercept to handle updates. The **UltraWinGrid** supports a whole host of events that you can subscribe to. Most of the names are pretty self-explanatory, and you can learn more about them from the Infragistics documentation. The one I want is **BeforeRowUpdate**, whose meaning should be pretty clear.

🗱 OpenEdge Editor - Samples/Updatable/Co	istomerUl	traGridUpd.cls - t	OpenEdge Archi	tect - C:\Gui4DotNel	t					-0×
File Edit Design Navigate Search Project	OpenEdge	Run Window H	telp							
] 😁 - 🔛 🛆 🔏 📖 🕲 🎋 - Ο	- 💁 -] 🛷 •] 🛬 •	8 - 🌣 🗢	• - ·						
😰 🤹 OpenEdge Editor										
🥰 Resources 🖾 🔅 🗖	C IMode	el.ds 🛛 🕄 Cust	omerModel.cls	🖏 *CustomerURra/	SridUpd.cls (Desig	n) 82				- D
k 😑 🗞 🏹									Toolbox	4
🗏 🐉 Samples 📃	2 0	ustomerUltraGri	dUpd						CustomControls	
Referenced Assemblies	•			Updatable Cust	omer Grid			-	Microsoft Contr	ola
B−JL Procedure Libraries				Opdatable Cust	Unier Grid			- 1	OpenEdge Cont	
TreeView	1	Text	Text	Test		R		- 11		
🗆 🍛 Updatable		Text	Test	Test		F		-11	🗄 OpenEdge Ultra	a Controls
CustomerModel.cls		T Grm	T G M	1 um		14		-11		
CustomerUltraGridUpd.cls										
C 1Model.ck										
🔠 Outine 💀 08 5 💷 Pro 28 🕒 🗖	0									
₩ 处 咫 ♡										
moUltraGridCustomer : Infragistics.Win.UltraWin 🕶										
Properties Events										
BeforeRowLayouttemResized	11							- 1		
BeforeRowRegionRemoved	1 14					_		- 11		
BeforeRowRegionScroll		Start! 6	ick 'Start!' to b							
BeforeRowRegionSize BeforeRowRegionSplit	0			0				-		
BeforeBowBesize										
BeforeRowsDeleted										
BeforeRowUpdate	•							•		
BeforeSelectChange										
BeforeSummaruDialog *	🤍 n	noBSCustomer								
About. UltraGrid Designer Lavout										
BeforeRowUpdate										
Occurs before a row is updated.	Cons	ole 🔝 Problems	🖾 🧟 Tasks						1	<u>> (</u>
1° B										
ju un							1			

Double-clicking on that event, I get a handler for it. And in the code I see it takes an event args subclass called **CancelableRowEventArgs**.

Let me find out what I can about that. In the **Class Browser**, I enter the name of the event args class, and look at its properties. There's a **Cancel** property, first of all, which is of type **Logical**, so I know that if I set it to **True**, then the update will be cancelled.

COPENED OPENED OF CONTRACT OF CONTRACT.	sGridUpd.cls - OpenEdge Architect - C:\Gui4DotNet	IOX
File Edit Source Navigate Search Project OpenEdge R	tun Window Help	
] 🔁 • 🗟 🛆 🔏 💷 🦦 🏇 • 🔕 • 💁 •	$\mathscr{G} \bullet [[] : \emptyset \circ \phi \bullet \phi$	
📑 👫 OpenEdge Editor		
Class Browser 23	수 - : 😽 😸 😣 😑 🔶 🏱 🗖	
Search CancelableRowEventArgs	Constructors	-
Infragistics.Win.UltraWinGrid.CancelableRowEventArg	GancelableRowEventArgs (Infragistics.Win.UltraWinGrid.UltraGridRow)	
	Methods Properties	
	- Concel	
	- G Next-Sbing	
	Row	.*/
	O Data Members	
	-& Empty	
	PUBLIC PROPERTY Cancel AS logical GET	
	SET	
	Nember of System.ComponentHodel.ConcelEventArgs	1. 11.
	Summary:	
		2
1° 10		
] U" (<u>B</u>)		

There's also a **Row** property, and if I drill down into that, and look through *its* properties, I see that it holds a reference to the **Band** in the grid where the selected row is.



And again, looking through the properties, I see a **Key** property.

Image: Construction Later Image: Constructint Later	e Edit Source Navigate Search Project OpenE		
Case Derword 13 Sech Sech C Unschedung C Unschedungehogene	😬 - 🔛 🛆 🔏 📖 🕲 🌣 - 💽 - 🍕	• A + 2 + 3 + 5 (p + + + +	
Otsat Browner 12 Image: Control Control Software Conthe Software Control Software Conthe Softwar	CoenEdge Editor		
ewch Image: State of Conversion of Conve			
Butsondand Butson Butsondand Butso			
Burgeddand B	sarch 📄	🕺 🔲 — 🚱 HeaderYisible	3
Guesstead Communication and a second se	IBraCridland		
Busicidand-figuration of the second figuration of the second figur			
G Unsödsdard TypeConverte Debriskorisug/Synthesisten G Unsödsdare Harcie Unsthesisten Debriskorisug/Synthesisten <td></td> <td>- us internation</td> <td></td>		- us internation	
C Ubwickster C Ubwic		- G IndentationGroupUyRow	
• Unspäcializations • Unspäcializatit		- G Indentation/woupby/tovic/pansionIndicator	
Uksäckdelingen och en son son son son son son son son son so			1.1
Dupsorch-Charlessbecker Dupsorch-Charlessbecker			11.1
• Uksändschlarvy • Uksändscharvy • Uksändstarvy •			-
• Utsyloid(chimory/ouner/do • Utsyloid(chipory/ouner/do • Utsyloid(chipory/ouner/do <td></td> <td></td> <td></td>			
• O throadschemonylohe • O throadschemonylo			
O thraddochemory.UtBrenet	G UltraGridCellProxyRole		
• Ubs/sid/damd • Ibs/sid/damd • Ubs/sid/damd • Ibs/sid/damd • Ubs/sid/damd/sized/samilyseCom • Ibs/sid/damd/sized/samilyseCom • Ubs/sid/combd/sized/samilyseCom • Ibs/sid/combd/sized/samilyseCom • Ubs/sid/combd/sized/samilyseCom • Ibs/sid/combd/samilyseCom • Ubs/sid/combd/samilyseCom • Ibs/sid/combd/samilyseComplyseComm • Ubs/sid/combd/samilyseCombd/	 UkraGridCellProxyUIElement 		
• Ubsyndodum • Ubsyndodum • Ubsyndodum • Ubsyndom • Ubsyndom • Ubsyndom • Ubsyndom • Ubsyndom • Ubsyndom • Ubsyndom • Ubsyndom • Dissyndom • Ubsyndodow • Dissyndow <td>- G UltraGridChildBand</td> <td></td> <td>1</td>	- G UltraGridChildBand		1
O Uhrwickikum Choose OET O Uhrwickikum Choose OET O Uhrwickikum Choose OET O Uhrwickikum Choose SET O Uhrwickikum Choose SET O Uhrwickikum Choose SET O Uhrwickikum Choose SET O Uhrwickikum Choose Nemary: O Uhrwickikum Choose Summary: <	UltraGridColumn	Next-story 2	9
O Brødschundboder OET O Brødschundboder OET O Brødschundboder ST O Brødschundboder ST O Brødschundboder Hember of Infragistics.Vin.UltraVinGrid.UltraGridBand O Unsødsdingholynov Hember of Infragistics.Vin.UltraVinGrid.UltraGridBand O Unsødsdingholynov The internally assigned key value for the band. The Key property is read- only for the Band object. O Unsødsdoup-fougtsyczifné Othersidscup flynoving			<u> </u>
• Ukrówskichobskic/ SET • Ukrówskichobskic/obedic/ober SET • Ukrówskichobskic/ober/ • Ukrówskichobskic/ Hember of Infragistics.Vin.UltraWinGrid.UltraGridUand • Ukrówskichopky Numary: Ikrówskichopky • Ukrówskichopky The internally assigned key value for the band. The Key property is read- only for the Band object. • Ukrówskichopky Ukrówskichopky • Ukrówskichopky Ikrówskichopky	UltraGridColumnChooser		
Of Unside/abs/lat/one/ Hember of Infragistics.Vin.UltraVinGrid.UltraGridBand Of Unside/abs/lat/one/ Hember of Infragistics.Vin.UltraVinGrid.UltraGridBand Of Unside/abs/lat/one Summary: Of Unside/abs/lat/one The internally assigned key value for the band. The Key property is read- only for the Band object. Of Unside/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/one/Houtlar/abs/lat/abs/la			
O thradidityphow The integration of finite property is read- only for the Band object. O thradidicapt HursdisconDypercement The integration of the band. The Key property is read- only for the Band object.			
Of Ubs/sidingh/Now Summary: Of Ubs/sidingh/Series The internally assigned key value for the band. The Key property is read-only for the Band object. Of Ubs/sidioup/ForupLayouline Ubs/sidioup/ForupLayouline Of Ubs/sidioup/ForupLayouline Ubs/sidioup/ForupLayouline		Member of Infragistics. Win, UltraWinGrid, UltraGridBand	
O Ubs/od/Bankaw The internally assigned key value for the band. The Key property is read- only for the Band object. Or Ubs/od/oupHoruplayoutho Ubs/od/oupHoruplayoutho Ubs/od/oupHoruplayoutho			
- O thraddown only for the Band object. - O thraddown for thraddown by the for the Band object. - O thraddown for the Band object.		Summary:	
G Ubradidioup+GroupLayoutinta G UbradidioupUHradidiauphpeConver			
Of UtraGridGroup-UtraGridGroupTypeConves		only for the Band object.	
- G UtzaGridGroupDyRow		-	
		wei -	
- 🕑 UtraGridLayout			
		-1	- E
- C UltraGindLayout+DisolaritanoutTypeConver	UltraGridLayout+DisolavLayoutTypeCon	Ner C	100

This is often used to hold the name of the data item that's displayed. For instance, the **Key** property of a **GridColumn** holds the name of the field displayed there, which can be useful. Here it's the name of the buffer whose fields are displayed in this band. That will be useful to me in my event handler.

So I've learned a little about how to manipulate the event args parameter to **BeforeRowUpdate**: I need that buffer name, because I want to be able to tell **SaveData** what buffer was updated, in case there's more than one. And I learned from the **Class Browser** how to drill down into the event args to get the **Row**, then the **Band** for the **Row**, and the **Key** value for the **Band**. That's the buffer name;

```
METHOD PRIVATE VOID moUltraGridCustomer_BeforeRowUpdate
( INPUT sender AS System.Object,
   INPUT e AS Infragistics.Win.UltraWinGrid.CancelableRowEventArgs ):
DEFINE VARIABLE cBuffer AS CHARACTER NO-UNDO.
DEFINE VARIABLE cCustName AS CHARACTER NO-UNDO.
cBuffer = e:Row:Band:Key.
```

There might be circumstances where an event fires when no actual changes were made to a row, so I check the binding source **RowModified** property:

IF moBSCustomer:RowModified THEN DO:

This property will be true if anything in the row was changed, and it will stay true as long as the grid is positioned to that row. Next comes the key step in the update through the binding source. When I invoke its **Assign** method, the changes in the grid – which you can think of as being like the screen buffer in older ABL terms – are transferred to the underlying record buffer, in this case, the temp-table row that the data came from. If there are any errors in that **Assign**, for instance if the UI let the user type in a value of the wrong data type into a cell, then the **Assign** will fail, so I check for an error return from **Assign**, set the **Cancel** property in the event args that I learned about in the **Class Browser**, and leave the update block:

```
IF NOT moBSCustomer:Assign() THEN
DO:
    /* Invalid data was entered in the grid row */
    e:Cancel = TRUE.
    LEAVE.
END.
```

But if the **Assign** succeeds, my changes have been moved to the temp-table, so that the code in the Model can validate them.

This part of the sequence is important. If there were update format masks you wanted to define in the UI itself to help determine what the user enters, to make sure that an integer is entered into an integer field, for instance, or to define a dropdown list of possible values, that's one thing, but you wouldn't want anything else in the way of business logic to execute in the View. In the case of my simple validation of **CustomerBirthCountry** being USA or Germany, I could have defined a grid cell dropdown that I then populated with those selection values. But ABL code that does calculations or validations that constitute real business logic doesn't belong there. My validation check in the **CustomerModel** class is really a placeholder for business logic that would normally execute on the server-side in a distributed application.

The key thing is that the **Assign** has transferred the updated values to the Model's temp-table, without the UI itself knowing anything about the underlying data storage, and the Model can then execute its validation without knowing anything about the UI. In any case, if **SaveData** returns an error, then I set **Cancel** just as before.

```
IF NOT moCustomerModel:SaveData(cBuffer) THEN
DO:
    e:Cancel = TRUE.
    LEAVE.
END.
```

I can now save this and run it, select a cell, make a change to it, and select another row. Anything I do to leave a row causes **BeforeRowUpdate** to fire, so my update has been executed. There's no visible confirmation of that, though, which I will address later.

🖹 💱 OpenEdge Editor							
Resources 33	Mo	del.cls CustomerModel	cls 🚳 CustomerUltra	GridUpd.cls (Design)	CustomerUltrat	GridUpd.ds 🕺	0
	204	Notes:					1
🗉 👺 Samples	205						*/
Referenced Assemblies	🛃 CustomerUltr	aGridUpd			_10 ×	1	
JL Procedure Libraries Frode	r					1	
B-B TreeView	CustomerFirsth		Ipdatable Customer Grid	C	A	stArgs):	
🗉 🍋 Updatable	Bobert	Lame CustomerLastName Kenned/Modified	CustomerBithCountry USA	CustomerBithdate 07/13/1979	CustomerGender	-	
CustomerModel.cls	0.1	Wagnet	Germany	09/14/1987		1	
CustomerUltraGridUpd.	Alce	Washington	USA	05/19/1982			
	Datab	Levis	Gemany	03/19/1970			
Cutine 📅 DB S 🖾 Pro Σ	Enic	Myers	Germany	06/08/1950	<u>v</u>		
E 1	Laura	Tores	Germany	07/16/1984			
Property Value	Carol	Murphy	Germany	12/01/1982			
🖃 Info		Jenkins		12/29/1954			
derivec false editabl true	Thomas		Germany	12/23/1954	9		
last mc January 14, 2010	Justin	Watson	USA		9		
linked false	Arthur	Tuner	USA	04/11/1980	V		
location C:\Gui4DotNet\Sa	Amanda	Roberts	USA	03/11/1971		1	
name CustomerUltraGrid path (Samples/Updatabl							
size 12,833 bytes							
	220	RETURN.					
	230	ALL DOWN					
	231	END METHOD.					
		*					

Next I change another name, and then tab to the **CustomerBirthCountry** cell, and enter an invalid value. When I leave the row, I see the error that came back from **SaveData**, which is checking the values that were assigned to the temp-table.

			-		1.00		
Resources 23	IModel.ds		ds 🗳 CustomerUitra	GridUpd.cls (Design)	CustomerUltraG	ndJpd.ds 🗙	
	204	Notes:					
🐸 Samples		A					····· ,
	🛃 CustomerUltraGrid	Upd					
Group Contraction Contractions			pdatable Customer Grid				
I Cover TreeView	CustomerFirstName	CustomerLastName	CustomerBithCountry	CustomerBitthdate	CustomerGender	ttkrgs):	
😑 🇁 Updatable	Robert	KennedsModified	USA	07/13/1979			
CustomerModel.cls	Dubu	WagnerChanged	France	09/14/1907			
CustomerUltraGridUpd	1 ×	sage (Press HELP to vie		05/19/1982			
	0.4.4			03/19/1970	<u> </u>		
Outline 🚰 DB S 💷 Pro S	Enc	walld Birth Country value, r	nust be USA or Germany.	06/08/1950	<u> </u>	1	
E 1	Laura	ОК	Help	07/16/1984		1	
operty Value	Carol			12/01/1982		1	
Info deriver false	Thomas	Jenkins	Germany	12/23/1954	R	1	
editabl true	Justin	Watson	USA	01/27/1983			
last mc January 14, 2010	Athur	Turer	USA	04/11/1900	- F	1	
Inked false location C:\Gui4DotNet\Sa	Amanda	Roberts	USA	03/11/1971			
name CustomerUltraGrid		Lumence		1	- <u>-</u>	1	
path /Samples/Updatabl						1	
size 12,833 bytes							
	220						
	229	RETURN.					
	230						
	231	END METHOD.					

By default the **Cancel** just erases my changes altogether, which is not very friendly, so that's something else I'll improve on before I'm done. I have at least confirmed that a basic update goes through, and that **SaveData** has written it back to the data source, the database table.

[The following part of the document corresponds to the second part of the video presentation.]

Since there are a few things that I don't like about how my sample grid behaves so far, I'm going to make some changes to show a few of the user interface options that can help improve the form's usability, as well as more of the ProBindingSource properties and methods.

One issue is that if I enter invalid data in a row, such as France for the CustomerBirthCountry name, I get an error message, but then lose my changes when the code sets the **Cancel** property. I could write some ABL code to make it behave differently, but if I select the grid in Visual Designer and look through its properties, I see that there is one called **RowUpdateCancelAction**, which looks like a perfect description of my situation. It's defined in terms of an enumeration, and if I drop down its list of values, I see that the default is **CancelUpdate**, which is what I saw happen, but the other choice is **RetainDataAndActivation**. That's exactly what I want to have happen: to leave the changes in place and leave me on the row with the error – that's what Activation means, that the row is still active – so that I can see and either correct the error or press Escape to cancel the changes myself.

🗑 OpenEdge Editor - Samples/Updatable/Cust	omerUltraGridUpd.cls - (IpenEdge Archite	ct - C:\Gui4DotNet					_IO ×
File Edit Design Navigate Search Project Op	enEdge Run Window H	ыþ						
10. B . Ku & \$.0.	0 1	5 6 6	-					
T CopenEdge Editor		4 4						
		1-	1					
🖏 Resources 🕴 🗖 🗖	CustomerModel.cls	IModel.cls	🔤 🌄 *CustomerUltra	vGridUpd.cls (Design)	22			- 0
						4	Toolbox	4
🖻 🚰 Samples 🔺	🛃 CustomerUltra	FridUpd					CustomControls	
Referenced Assembles	Q		Updatable Cur					
B-JL Procedure Libraries				_	Microsoft Contr	rols		
🖲 😂 rcode			Text				OpenEdge Con	alorte
E Dev TreeView	Text	Text	☑ _/_/		DpenEdge Ultr	a Controls		
🖻 😂 Updatable	Text	Text	Text		E _/_/			
CustomerModel.cls								
CustomerUltraGridUpd.cls								
R IModel.ds								
🚼 Outine 🐖 DE Str 💷 Prope 🗶 🖓 🗖	0							
₩ 24 🖾 🏹	1							
moUltraGridCustomer : Infragistics:Win.UltraWinGr								
Properties Events								
MaximumSize 0,0								
E MinimumSize 0,0					_			
Modifiers Private	Start!							
RightToLeft No	0		0					
RowUpdateCancelAction RetainDataA ··								
E Size 602, 283								
StyleLibraryName						× 1		
StyleSetName SyncWithCurrencyManag True	•				•	· 1		
Tabindex 0								
TabStop True	moBSCustomer							
About, UltraGrid Designer, Layout Wizard								
RowUpdateCanceAction						_		
Specifies the action that should be taken when	🕒 Console 🔝 Problem	ns 🕮 🧭 Tasks						<u>⊳ ⊽ ⊓ □</u>
] 🗗 🗈								

This is a perfect example of both the value of these .NET controls, especially the UltraControls, and also the challenge of using them: On the one hand, there are many properties, methods, and events to sort through to find what you need to solve a problem. On the other hand, there's almost always built-in support for what you need the control to do, so it's worth your while getting to know the controls through the documentation and just reviewing the names of everything the controls support, which are usually clear in telling you what they do. Naming conventions are consistent enough that with a little experience you will get a sense for what to expect and what to look for in a control that's new to you.

If I run the form again with **RowUpdateCancelAction** changed to **RetainDataAndActivation**, I can make a change to a Customer, and then change the CustomerBirthCountry to an invalid value. Now I see the error message as I did before, but my changes haven't been canceled:

DpenEdgeEditor - Sar le Edt Design Naviga					Architect - C:\Gui4Dot/	iet.			
1 · 1 A A					· • • • •				
OpenEdge Editor									
Resources 23		- 0	Customer	Model.ds 💽 IMod	lel.ds 😽 Customert		×		- 6
		28 7						Toolbox	4
🖻 🎒 Samples			🔐 Custo	omerUltraGridUpd				CustomControls	-
🗄 🧬 Referenced As		🛃 Custom	ertiltraGridU	pd			LOX	Hicrosoft Controls	-
IL Procedure Libra	aries .								-
🖲 🇁 rcode					pdatable Customer Grid			OpenEdge Controls	_
🖲 😂 TreeView 🖻 🍋 Updatable			erFisiName	CustomerLastName	CustomerBirthCountry	CustomerBirthdate	CustomerGender	🗈 OpenEdge Ultra Contr	ol
Updatable	and all offer	Robert		KennedyModified	USA	07/13/1979			
CustomerU		9 Peter		WagnerModified	France	09/14/1987	R		
1Model.ck	a do seren	Alice		Washington	USA	05/19/1902			
Cutine 📅 DE Str	C Deces	Patrick		Lewis	Germany	03/19/1970	2		
Coone Hill on to		Eric		Myera	Gemany	06/08/1950	R		
	8	Laza		Torres	Gemany	07/16/1984	<u> </u>		
SustomerUltraGridUpd : Pro	gress.Windo	Cwol		Murphy	Gemany	12/01/1992			
Properties Events		Thoma		Jenking	Gemany	12/29/1954	R		
Opacity	100%	Justin		Watton	USA	01/27/1983	<u> </u>		
E Padding	0, 0, 0, 0				USA				
previousState	Normal	Arthur		Tuner		04/11/1980	9		
RightToLeft RightToLeftLavout	False	Amand	8	Roberts	USA	03/11/1971	- J		
Showlcon	True								
ShowInTaskbar	True						t t		
3 Size	614, 36_						f	1	
SizeGripStyle	Auto		<u> </u>						
StartPosition	WindowsDv	elault.c	10 mB	SCustomer					
Tag Test	Customer	-	1						
Test	Ealte	URIAG +							
Text The text associated with th			Console	🖹 Problems 🛙 🍝	Tasks				-
19 🖹					Writ	sble			

The row in error is still the current row, as indicated by the row edit icon over on the left, so I can click in the cell where the error is, and correct it, and then move out of the row and save all my changes. That's much more user friendly. I try another row, and once again enter an invalid CustomerBirthCountry, and once again see the error message. It's a characteristic of the UltraGrid that I can press **Escape** to cancel an update, so I have that option to go through with the Cancel myself.

I present this as an example of how you can expect that the .NET controls will provide countless variants of behavior that you can capture and take advantage of. You just need to be thorough enough to find and use them.

The next issue I had with my updatable grid was that there was no visible feedback when an update succeeded. As one example of how I can deal with that, I'll add a status bar to the form to display an update status. Here among the Ultra controls is a **StatusBar** that I can drop onto the bottom of the form.

: Edit Design Naviga 📸 = 🔛 🗀 🗌 🔏 L											
CoenEdge Editor											
Resources 33		10	Custon	rerModel.ds	Model.ds	Sustoner	RraGridUpd.cls (Desig	0.52			•
		7			0.000					Toolbox	
🛛 📁 Samples		a I	🛃 Cu	stomerUltra						222 UltraLis/View	
Referenced As	sembles	1								100 UltraMaskedEdt	
-JL Procedure Libra	ries -					Updatable	Customer Grid			28 UltraMonthViewMulti	
🗄 🗁 rcode										50 UltraMonth/iewSingle	
🕀 🗁 TreeView			► T	eof	Text	Text	_/_/	₽.	/_/	UltraNavigationBar	
🖻 😂 Updatable			Te	ext	Text	Text		R.	7.7	UltraNavigationBar	
- CustomerM		11						1.65 1-		1 UltraNumericEditor	
CustomerU	traGridUpd.ds	U 1								UltraOptionSet	
Model.ds		1								UltraPanel	
Outine 📅 DB Str	Prope., 23	וונ								UltraPictureBox	
000000										UltraPopupControlContai	ner
	te ĝ↓ 03 5									UltraPrintDocument	
saStatusBar1 : Infragistics	Win UltraWinStatusB									UltraPrintPreviewControl	
roperties Events	_	1								UltraPrintPreviewDialog	
		a 1								C UltraPrintPreviewThumb	hai
(About)	-									UltraProgressBar	
(DataBindings) [Name]	ultraStaturflar1			Start!	Click 'Starti' to b					S UltraSchedulePrintDocu	ner
AccessibleDescription	ukiastakusbali			JUGHU	Citor Start 1010	egn designing (UltraScrolBar	
AccessibleName							0			UltraSpellChecker	
AccessibleRole	Default		ubaStr	alusBar1			0			UltraStatusBar UltraTabbedMdManage	
AllowOrop	False	11								UltraTabControl	1
AlphaBlendHode	Optimized	1.1							•	UltraTabStripControl	
Anchor	Top, Left	14	_							UltraTabStipControl	
Appearance				BSCustomer						UltraTimeZoneEditor	
Appearances	(Collection)		·* mo	AL2 CONTOUNED						UltraT oobarsManager	
bout, Custom Property Pa	- 1991									TibuToolTioManager	
Name)										4	2
										16 🗸	

I rename it to be consistent with the other control names I've used. Without going into all the rest of its properties, I initialize its **Text** property to "Status: ", to make that the value initially displayed.

	🗑 OpenEdge Editor - Sa					ct - C:\Gui4Dot	Net				
Control Product Start Start Start Control Product Start Start Control Product Start	File Edit Design Naviga	te Search Project O	enEdge	Run Window H	elp						
# Mexaces II Image: Contrament/cold.(ik PhodeLik P	😂 • 🔛 🛆 🔏 ا	😀 🗞 i 🌾 i 🖓 i	Q] 🛷 •] 🔄 •	11 - 💝 🔶 -	$\rightarrow \tau$					
Windows S Consent Out Subject (Source View) / Subject (S	📑 💐 OpenEdge Editor										
Supplies Supplie Supplies S	🖑 Resources 💠	- 0	C CL	stomerModel.cls	Model.cls	Custome	rUltraGridUpd.cls (Design) 22			- 0
Image: State Processor data state data Image: State Processor data		- 🕂 🗑 🕒 🛸 🏱								Toolbox	4
B defense dataset data			🖪	CustomerUltraG	ridUpd						-
Image: State in the control Image: State in the control <td>B 📌 Referenced As</td> <td>sembles</td> <td>-</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td>	B 📌 Referenced As	sembles	-								
Image: Statute Image		aries —				Updatab	le Customer Grid				
Text											
Ford Text Text Classic hogo and a constrained of					Test	Test					
Control C				Text	Text	Text		V			
Decked de			1 1			-		-			
Course @ Del 20 Proster, 12 Del 20 Proster, 12 Del 20 Del		RraGridUpd.cls									
Counter (20 to 2000) The count of the count of the counter of the											
Image: State in the state	Cutine PD8 Str	Prope 23 - 0									
Implementation Implementation Standball of on molthet Syndball of on molthet Standball Standball <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>UltraPopupControlContain</td> <td>er</td>										UltraPopupControlContain	er
Indexted at imagines introduced at imagines into at imagines introduced at											
Progetic Events Standfail for molthet Standfail for molthet Sydschlawer Standfail for molthet Sydschlawer Standfail for molthet Sydschlawer Ubschlawer Takings I Test I Understring Ubschlawer Understring Ubschlawer Ubschlawer	moUltraStatusBar : Infragist	ics.Win.UltraWinStatus 💌									
Statistic of on notifie Synkickapitalise Synkickapitalise <td>Properties Events</td> <td></td>	Properties Events										
Sydicitions I Takinos I Takino	StatusBarTest on moU	be 🔺		.1							-
SyderStreve Tablobe I Start I Cick Start be begin designing the Utradoid Ut	StyleLibraryName	_	L 14					_			ert
Takinos I Takinos I Takinos Stanna Testinos Stanna Testinos Stanna Testinos Dista Umargotive Dis	StyleSetName			Start							
Top Statu:			1 1					_			
Intel Statut Ubit al docted did did did did did did did did did d		False					0			UltraStatusBar	
Teellenderupflade Orland Usefactivityer Usefactivityer Usefactivityer Usefactivityer Washermonic Tuse Tuse Tus			St	30us:					_	UltraTabbedMdManager	
UnderSchleig Tune UnderSchleig Tune UnderWenneis Tune Underwenneis Tune Water Control Paters Tet Te tet associeted with the control Dessele © Problems 32 @ Tasks									*	UltraTabControl	
Usef and the set of th			-							UltraTabStripControl	
Underworking True WindSClusterer WindSclusterer Under Confector East. About Custom Process Pase Test Test Tor Test Test Confector East Conside with the control Conside WindSclusterer East Conside Sclusterer Ea											
Adout Cuttom Procests Pares IIII Uhal TobburthKrauger Text IIIII Intel root Function The toot associated with the control IIIIIIII Intel root Function			4	moBSDustomer							_
Test The test associated with the control Denselle 🐍 Problems 32 🖉 Tasks 😵 🕫 🗖											
The lest associated with the control		8085.u								TithaTooTinManager	اڭ ر
		he control.	0 Ce	ansole 💽 Problem	s 22 🖉 Tasks					<u>*</u> *	
										47	
	. 📬 🔃										

Now I need to add a couple of lines of code to assign a value to display after an update. At the beginning of the **BeforeRowUpdate** event handler. I reset the **Text** property back to its initial value, so that it's reset each time an update happens:

DEFINE	VARIABLE	cBuffer	AS	CHARACTER	NO-UNDO.
DEFINE	VARIABLE	cCustName	AS	CHARACTER	NO-UNDO.
cBu	uffer = <mark>e</mark> :	ROW:Band:F	Key.		
mot	JltraStatu	sBar:TEXT	= '	'Status: ".	

Then at the end of the method I construct a string that displays the first and last name fields:

cCustName =	e:ROW:Cells:Item[0]:Text + " " +
	e:ROW:Cells:Item[1]:Text.

This is worth taking a look at for a moment. Remember that the method's **CancelableRowEventArgs** parameter has a **Row** property. The **Row** contains a **Cells** property, which is a .NET collection of all the

cells in the row. The standard way to access an element of a collection is with the **Item** property, which takes a zero-based index to the cell you want. The **Text** property of each cell is the value it contains. So **Item[0]** and **Item[1]** are the first two columns in the grid, the CustomerFirstName and CustomerLastName fields.

To see how my status bar looks, I can rerun the form, click on a row, change the name, and change the CustomerBirthCountry to a valid value. When I select another row, the **BeforeRowUpdate** handler fires, and I see the status message.



So this is just a simple example of integrating another Ultra control into a form. While I'm running the form and changing values, I can show you another aspect of the update that you could consider an issue. If I click on the CustomerLastName column header, the **SortData** method that I coded in the presentation series on sorting data with the ProBindingSource re-opens the temp-table query, sorted by CustomerLastName. Now if I select a customer, and change the CustomerLastName to a value that should sort somewhere else, the change is saved, but the data isn't resorted, because I didn't ask to reopen the query.

We Edit Source Navigate Search								
<u>□• □ ⇔ ∡ </u>	2	0 • % •] //	•] 8 • 8 • 4	· 🗢 • • • ·				
🝸 🎳 OpenEdge Editor				-				
Resources 23		Customer		el.cls Sustoment	RraGridUpd.cls (Design)		Jupd.ds 🗙	- 1
				r AS System.Obje		Drekowopdate		-
🖻 🎒 Samples		221				CancelableRowEyen	tArgs):	
Referenced Assemblies	•	CustomerUltraGridU	pd					
B-JL Procedure Libraries	_							
Code TreeView		-		stable Customer Grid				
E Codatable		CustomerFirstName	CustomerLastName /	CustomerBirthCountry	CustomerBithdate	CustomerGender		
CustomerModel.cls		Patrick	Alexander	semany	02/08/1983			
CustomerUltraGridUpd.c		Robert	CharlieBaker	JSA	04/29/1974	V		
Model.cls	Þ	Susan	Barnes	iermany	07/17/1972			
🗄 Outline 🕴 🚰 DB Str 🗔 Prop		Evelyn	Bing	USA	12/10/1965			
		Helen	Brooks	Germany	07/31/1974			
USING Declarations		Lany	Brooks	USA	02/29/1972	V		
		May	Burns	USA	09/02/1956			
Methods		Melissa	Bums	Germany	10/25/1959			
CustomerUltraGridUpd		Richard	Burns	Germany	01/05/1968	2		
- CustomerUltraGridUpd		Frank	Burns	USA	04/10/1984	<u> </u>		
- Q InitializeComponent		Helen	Carter	USA	05/25/1951			
Contraction and the second contraction of th	14		I care	1034	103/23/1331			
moUltraGridCustomer_Befor	-							
	Stal	tus: Robert CharlieBaker	update successful.					
		2.43		OW:Cells:Item[1]	:Text + " updat	Le successful.".		
		2.4.4	END.					
		245						
		246	RETURN.					
4		Console	Rroblems 23 🥥	Tasks				10 V U
								-

Showing you how to make the resort happen will illustrate another part of the interaction between the user interface, the binding source, and the underlying data. Back in the **CustomerModel** class, I want to make a change to the **SaveData** method. If the save operation succeeds, I want to re-open the current query,

with whatever sort order is defined for it. That way, the data will be resorted properly if I make a change that affects a row's sort sequence in the grid.

```
/* If we get here all client-side validation succeeded. */
IF hBeforeBuffer:SAVE-ROW-CHANGES () THEN
DO:
    hBeforeBuffer:ACCEPT-ROW-CHANGES ().
    httCustQuery:QUERY-OPEN ().
    RETURN TRUE.
END.
```

Looking at the ProBindingSource properties again makes it clear what's making the resynchronization of data between ProBindingSource and the UI work. The property that plays a key role here is **AutoSync**.

🔃 OpenEdge Editor - Samples/Updatable,			chitect - C:\Gui4	iDotNet				_ 🗆 ×
File Edit Design Navigate Search Project	t OpenEdge Run Wind	low Help						
] 🗈 • 🗟 📥] 🔏 📖 🦦) 🌼 • •	ا • 94 • 🛛 🖉 • 🖉	5 · 5 · 6	$\Leftrightarrow \bullet \Rightarrow \bullet$					
🞦 👫 OpenEdge Editor								
🦉 Resources 😫 🔷 🗖	CustomerModel.cls	IModel.cls	CustomerU	ltraGridUpd.cls (Desigr	n) 🖾 💽 Custon	erUltraGridUpd.	ds	- 0
🙀 😑 📚 🏹							Toolbox	4
🖻 🎬 Samples 📃	🛃 EustomerUltra	GridUpd	_				CustomControl	
Referenced Assemblies T_JL Procedure Libraries			Updatable	Customer Grid			Microsoft Cont	rols
							OpenEdge Cor	strols
TreeView	Text	Text	Text					
Updatable CustomerModel.cls	Text	Test	Test	_/_/		/_	- opencoge on	a controls
Cott	Customer 1		xegin designing	the UltraGrid				
	Console 🖹 Proble	ms 🖾 🙋 Tasks						• • • •
J 📭 🖪						1		

It's **True** by default, which is what supports the behavior I've been showing you. If **AutoSync** is **True**, then any time you re-open a query bound to a ProBindingSource, or use one of the ABL **REPOSITION** statements to change the selected row in the query, the binding source **Position** property is reset automatically, and any user interface controls bound to the binding source are also refreshed to stay in sync with the data. This is probably the behavior you want almost all the time, but if you ever want to control when data gets refreshed yourself, you can set **AutoSync** to **False** and use the binding source **Refresh** method.

To take a look at the result of the query re-open, I save and re-run the form, re-sort the data by CustomerLastName, select a row, and make a change that changes its sort position.

Resources 🕅	2		CustomerModel.	ds 💽 IModel.ds	🖏 CustomerUltraGrid	Upd.ds (Design) 23	CustomerUltraGridUpd.	ds
	> 👳 📄 😫	0	,					Toolbox
🗆 🐸 Samples		i -1	EustomerUltraGridU	pd			_IO × -	CustomControls
	nced Assemblies				pdatable Customer Grid			Microsoft Controls
B-JL Proced	ure Libraries		CustomerFirstName	CustomerLastName 4	CustomerBithCountry	CustomerBirthdate	CustomerGender	OpenEdge Controls
It is roode	-		Patrick	Alexander	Gemany	02/08/1983		
E 😂 Updata			Bobert	Baker	USA	04/29/1974		OpenEdge Ultra Control
	stomerModel.cls		Evelun	Bing	USA	12/10/1965		
Cu:	stomerUltraGridUpd.ck		Helen	Brooks	Gemany	07/31/1974		
		F	Lany	Brooks	USA	02/29/1972	2	
E Out 🔛 DB	🖿 Pro 🗙 🥈		Mary	Burns	USA	09/02/1956		
	田会に		Melissa	Burns	Germany	10/25/1959		
noBSCustomer : Pr	ogress.Data.BindingSc	F	Richard	Burns	Germany	01/05/1968	<u> </u>	
Properties Ev	ents		Frank.	Burns	USA	04/10/1984	<u> </u>	
(Name)	moBSCustomer		Helen	Cater	USA	05/25/1951	0	
AllowEdit. AllowNew	True		Susan	CCBarnes	Germany	07/17/1972		
AllowNew	True	1.	1	-				
AutoSort	Falce							
AutoSync	True	Sta	tus: Susan CCBarnes up	idate successful.			de.	
AutoUpdate	Falce							
DataMember	rane	- 11	L				F	
		-	2					
Batching DataMember	Falte	•	The second secon				<u> </u>	

Leaving the updated row, the query is re-opened and all the data re-synchronized with the grid.

Lastly, I'll show you a simple example of using the ProBindingSource **Refresh** method. To do that, I need a new method in the model, so I first add it to the interface. The new method, FormatColumn, takes a column name and its value as parameters, and re-formats the value under certain rules. If the method returns true, then the value has been re-formatted and needs to be re-displayed. That's where **Refresh** comes in.

ſ	INTERFACE Updatable.IModel:
	METHOD PUBLIC VOID FetchData (INPUT pcFilter AS CHARACTER).
	METHOD PUBLIC VOID SortData (INPUT pcSort AS CHARACTER).
	METHOD PUBLIC HANDLE GetQuery().
	METHOD PUBLIC LOGICAL SaveData(INPUT pcBufferName AS CHARACTER).
	METHOD PUBLIC LOGICAL FormatColumn (INPUT pcColumnName AS CHARACTER,
	INPUT pcColumnValue AS CHARACTER).
	END INTERFACE.

From the Source menu, I add the skeleton code for the new method:

	tomerModel.cls - OpenEdge Architect - C:\Gui4DotNet		_02
File Edit Source Navigate Search Project			
] 🗂 • 🗟 🗋 🔏 📖 🕲] 🏶 • 🕗	- 🗛 🔐 Override/Implement Members 📃 🗆 🗙		
🗈 🎳 OpenEdge Editor	Generate stubs to override or implement members		
🕰 Resources 💱 📃 🗌	Check inherited members that you want to override and members implemented via interface for which you want to generate stubs.	ustomerUltraGridUpd.cls	
Contraction of the set of th	Select members to override or implement: Select All Devided in Production Production Select All Select All Select All Devided All Select All Devided All Select All	ucceeded. */	1
	94 95 96 98 98 90		
	100 La		
	105 Select which style of member code you would like to create: 106 Image: Comparison of the second state of the	CTER):	
	C Generate default values for required members Of	NIT	
	Generate Cancel		
) 🕈 🖻	Writable Insert 95:25		

This is the code for the method:



If the CustomerBirthCountry column has a value of "usa" with any type of capitalization, the code for that case forces it to be all upper case, and returns **TRUE** to signal that the data value has been changed. Remember that because this is code in the Model, it's the temp-table value I'm changing, which has no direct effect on the UI.

So back in the form I need to invoke **FormatColumn** and check its return value. Looking again through the **UltraGrid** events, I find one named **AfterCellUpdate** that fires after a cell has been updated. A little experience will teach you where to look in these very full-featured controls for support for the behavior you need. The event handler for **AfterCellUpdate** takes an event args class of **CellEventArgs**, and if I were to look that up in the Class Browser, I'd see that, not surprisingly, it has a **Cell** property. The **Column** property of the **Cell** points to the column the cell is in. I have used the Key property once before to get at the buffer name for a band in the grid. Here the **Key** property holds the column name, so that becomes the first parameter to **FormatColumn**. The cell also has a **Text** property, which holds the column value, so that becomes the second parameter. And if **FormatColumn** returns **True**, it has modified the field's value in the temp-table:

```
METHOD PRIVATE VOID moUltraGridCustomer_AfterCellUpdate(
   INPUT sender AS System.Object,
   INPUT e AS Infragistics.Win.UltraWinGrid.CellEventArgs ):
    IF moCustomerModel:FormatColumn(INPUT e:Cell:Column:Key,
        INPUT e:Cell:Text) THEN
        moBSCustomer:Refresh().
        RETURN.
END METHOD.
```

This is where I invoke the ProBindingSource **Refresh** method to redisplay the row. Because the code hasn't re-opened the query or re-positioned it, I have to do the **Refresh** myself.

To test this latest change, I save and re-run the form, select a customer, change the CustomerLastName, and then change the CustomerBirthCountry to "usa", without using all capitals:

Resources II		- CustomerMod	N.ck Nodel.ck	🕹 CustomerUltraG	idt to d als (Provine)	CustomerLitraGrid.pd.ck	
			InitializeCom		uncharge (needlig		1
E 14 Samples	- + A 🛛 🖻 🕯	32		iel = NEV Updata	ble.CustomerMod	iel().	
Samples E P Reference	and Assemblies	Eustomert/tradicid				- O ×	
I JL Procedu		Costonieroitragnat	40				
🗄 🗁 rcode			U	pdatable Customer Grid			
B De TreeViev		CustomerFirstName	CustomerLastName	CustomerBithCountry	CustomerBirthdate	CustomerGender +	
🖯 😂 Updatab	le xmerModel.ds	Robert	Kennedy	USA	07/13/1979		
	omerModel.ds omerUitraGridUpd.r	1g Peter	WagnerChanged	Germany	09/14/1907		
C IMoc		Alice	Washington	USA	05/19/1902		
E out 🐖 ce	- Pro 22	Patrick.	Lewiston	USA	03/19/1970	3	
- oatin Ell con		Eric	Myers	Gemany	06/08/1950	V	
	Value	Laura	Torres	Gemany	07/16/1984		
ABL Type	Yarue	Carol	Murphy	Gemany	12/01/1982		
File Name		Thomas	Jenkina	Gernary	12/28/1954	2	
Length	1253	Justin	Watson	USA	01/27/1903	v	
Problems Scoped-to Node		Arthur	Tuner	USA	04/11/1900	8	
Start Position	2240	Amanda	Roberts	USA	03/11/1971	100	*/
	0	11					
	148						
Token Start Posi Token Type	-1 METHOD-DECLAR.	Status:				1	
Token Type	PETROD-DECLARG	188	TE moCustome	Bodel:FormatCol	INCOMENTATION	Li Column / You	
		189		litext) THEN	unitiment ercer	arcorumnikey,	
		190		omer:Refresh().			
		191	RETURN.				

When I leave the cell, the **AfterCellUpdate** event fires to run the **FormatColumn** method and refresh the displayed value. That's the last of the changes I'll make to this demonstration of the basics of updating data using the ProBindingSource.

Let's quickly review a few of the things I've shown you in this two-part session:

- Use the **TRACKING-CHANGES** property of a ProDataSet to allow you to **FILL** the DataSet and then keep track of changes to its contents.
- The save-row-changes method saves changes back to the source database table. ACCEPT-ROW-CHANGES marks those changes as being accepted in the dataset itself, and REJECT-ROW-CHANGES removes them.
- In the ProBindingSource, the AllowEdit property lets you manage whether UI controls that are bound to it are enabled for input or not.
- The AutoUpdate property is there just to help you do quick testing of updating data, but should be left False for serious development.
- Use the ProBindingSource RowModified property to check whether a row in the UI has actually been changed, and the Assign method to write changes back to the buffer in the Model that is the binding source's data source.
- Remember that the binding source AutoSync property allows an automatic refresh of data when the underlying query is re-opened or repositioned.
- When you need to synchronize a change that doesn't reposition the query, use the ProBindingSource **Refresh** method to push the change out to the UI.
- Use control events like BeforeRowUpdate and AfterCellUpdate to capture changes that you need to write back to the Model.
- Learn about useful control properties like RowUpdateCancelAction to take advantage of built-in behavior that you want in your user interface.

That concludes this two-part session on managing data updates with the ProBindingSource.