BATCHING DATA WITH THE PROBINDINGSOURCE

John Sadd Fellow and OpenEdge Evangelist Document Version 1.0 March 2010





PROGRESS SOFTWARE

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 video presentation that is part of the series on Using Visual Designer and GUI for .NET, and builds on the sessions that showed how to add a ProBindingSource to a form and then display and sort data in the user interface. In this session I extend those examples to show how to retrieve data for display in batches using a property and an event that are part of the ProBindingSource.

I start with the form I created in the sessions on data sorting, **CustomerUltraGrid.cls**, and a model class to manage customer data, **CustomerModel.cls**, that implements an interface called **IModel.cls**.

First take a look at the properties of the form's ProBindingSource. There's a **Batching** property that tells the binding source whether you want it to coordinate data batching or not. It's **False** by default, but for this example it needs to be set to **True**.

🚼 OpenEdge Editor - Samples/EustomerUltr	aGrid.cls - OpenEdge Ar						_IO X
File Edit Design Navigate Search Project	OpenEdge Run Windov	v Help					
😁 • 🛛 🛆 🛛 🔏 🗠 🗞 • 🔾	- 💁 - 🛛 🛷 - 🗋 🔮	- E - 🗢 🗢	•				
😭 🎳 OpenEdge Editor							
Resources 23	C IModel.ds	CustomerModel.cls	CustomerUltraGrid.d	ls (Design) 22	CustomerUltraGrid.cls		
					1	Toolbox	9
😑 🤔 Samples 🔺	🛃 CustomerUltra	aGrid			_0;	+ CustomControls	
Referenced Assemblies			Customer grid			+ Microsoft Controls	
B-JL Procedure Libraries			Contrainer grid		r r -		
Code TreeView	Text	Text	Test	/_/		OpenEdge Contro	
B- Collectable	Text	Test				🗄 OpenEdge Ultra 🕻	Controls
CustomerModel.cls	lex	1 ext	1 ext		M _/_/		
CustomerUltraGridUpd.cls							
Model.ds							
🚼 Outline 📅 DB 5 💷 Prop 🗶 🖓 🗖							
¥ 4↓ 113 ▽							
moBSCustomer : Progress Data BindingSource 💌							
Properties Events							
(Name) moBSCustomer A	1.1						
AllowEdit True		_			,		
AlowNew True AlowRemove True	Start!	Click 'Start!' to	begin designing the Ult				
AllowFiemove True AutoSoft False							
AutoSvnc True	Next						
AutoUpdate False							
Batching False •							
DataMember True							
False							
ProBindingSource Da	•				•		
	0	D .					
Batching Indicates whether the BindingSource will	woBSCustomer						
send an OffEnd event when a bound control.							
	📮 Console 🕴 🚺 Pr	oblems 🧟 Tasks			🗏 🗶 🙀 🕞 🌆	🖓 🖗 😁 🖬 •	📑 • ° 🗇 🗎
J 📭 🗈							

Basically all that happens when I set **Batching** to **True** is that the binding source will now fire its **OffEnd** event when it detects that the user has scrolled or otherwise gotten to the end of the data that's managed through the binding source.

Next we need to look at the binding source events. If I double-click on the **OffEnd** event, I get a skeleton event handler method. Before writing any code, we can look and see what event args class gets passed in to this event handler. Not surprisingly, it's called **OffEndEventArgs**.

			nEdge Architect - C:\Gui4DotN	rt.					- 🗆 ×
	Navigate Search Project (
) 📬 • 🔛 🎰)	1 8 5.0	- 🏊 - 🛛 🦽	2 - 4 - 6 - 6 🗇 -	÷ •					
🔡 🎳 OpenEdge E									
🖏 Resources 🕮	- 0	Model.ds	CustomerModel.cls	🖏 *CustomerUltra	Grid.cls (Design)	Custor			- 0
		44							*
E 12 Samples		45	END CONSTRUCTOR.						
E - P Referer	verl Assemblies	46							
B-JL Procedu		47							
🖲 🇁 rcode		40*	METHOD PRIVATE VOID	InitializeCo	mponent()	:0			
TreeVier	~	101							
🖻 🗁 Updatal		102							
	tomerModel.cls	100							
	tomerUltraGridUpd.cls	184							
- C 1Mo		185							
De a alus Into an		186							
92 Ongue All DR	S Prop 🕄 🗖 🗖	187							
	16 🋸 🗷 📍	188	/*						
Property	Value	189	Purpose:						
😑 Info		190	Notes:						
derived	false	191	· · · · · · · · · · · · · · · · · · ·						-*/
editable	brue	192	8VisualDesigner.						
	January 12, 2010 1:37:0	1930	METHOD PRIVATE VOID				AS System.	Object,	_
	false	194	INPUT args AS Progre	ss.Data.OffE	ndEventArgs):			- 10
	C:\GuHDotNet\Samples\d	195							
	dsCustomer.i	196	RETURN.						
	/Samples/dsCustomer.i	197							
size	987 bytes	198	END METHOD.						
		199							
		200							
		201	/*						
		202	Purpose:						
		203	Notes:						- Ľ
		1						_	-
		Console 2	🖾 Problems 🧔 Tasks			🗏 X 🕅	🕞 🚮 🖉 🖉	🖻 😁 • 📬	•
1 1 1				Witable	Insert	194 : 5			

Looking at that in the Class Browser, you can see that the only meaningful property is called **RowsAdded**. This needs to be set to tell the binding source how many rows have been added to the result set it manages each time the **OffEnd** event fires.

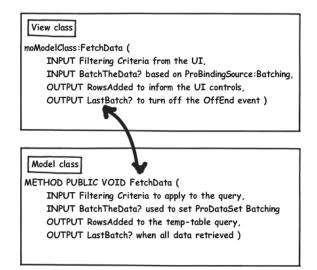
DpenEdge Editor - Samples/Customert/ItraGrid.cls - 0	penEdge Architect - C:\Gui4DotNet		
File Edit Source Navigate Search Project OpenEdge			
🗂 • 🔜 🛆 🔏 📖 🗞 • 🐼 • 🗛 •	A • 2 • 3 • 2 • 2 • • • •		
📑 🎳 OpenEdge Editor			
Class Browser 23			- 0
Search offendervertangs 🗾 🗶	Equals (System.Object, System.Object) Finalsen () Generation () Gen	A	•
	CLASS OffEndEventArgs : Member of Progress.Data Inherits System.EventArgs Summary: • VSING USING Progress.Data.OffEndEventArgs.	<u>ज</u> ज ज ज	-*/ *
) 🕫 🖪		J	

Adding support for batching requires adding a few parameters to the method definition in the interface for FetchData.

The first new requirement is that the View needs to tell the Model whether batching has been enabled, since the ProBindingSource with the **Batching** property is part of the View. This may not always be the right design for a completed application, because deciding whether data should be retrieved in batches is really part of data management, not user interface. So maybe the Model should be telling the View whether to enable batching in the binding source. But for this simplified example, the caller in the View will just pass in the **Batching** property value from the binding source down to the Model.

And as we just saw, the Model needs to pass back the number of rows that were added to the result set, so that the **OffEnd** event handler can set that value in the event args.

And finally, the Model needs to pass back another flag when the last batch of data has been retrieved so that the View can turn off **Batching** and the **OffEnd** event will stop firing. These changes are illustrated in this diagram:



And these changes to the **FetchData** method definition in **IModel.cls** need to be implemented by any model class that implements IModel:

```
INTERFACE IModel:
METHOD PUBLIC VOID FetchData (INPUT pcFilter AS CHARACTER,
INPUT plBatching AS LOGICAL,
OUTPUT piRowsAdded AS INTEGER,
OUTPUT plLastBatch AS LOGICAL ).
METHOD PUBLIC VOID SortData (INPUT pcSort AS CHARACTER ).
METHOD PUBLIC HANDLE GetQuery().
END INTERFACE.
```

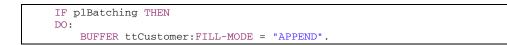
In the sample model class CustomerModel.cls I first add those parameters to FetchData:

```
METHOD PUBLIC VOID FetchData( INPUT pcFilter AS CHARACTER,
INPUT plBatching AS LOGICAL,
OUTPUT piRowsAdded AS INTEGER,
OUTPUT plLastBatch AS LOGICAL ):
```

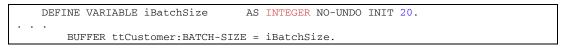
Remember that there's no separation between client-side Model and business objects running on an AppServer here; all that support is in my one simplified Model class. So after **FetchData** has prepared the database query based on whatever filtering criteria were passed in, it **FILLs** the ProDataSet and opens the query on its one temp-table, which the binding source in the View is connected to. Here's the first section of code in FetchData that needs to be extended to support batching.

```
cPrepare = "FOR EACH AutoEdge.Customer".
IF pcFilter NE "" THEN
    cPrepare = cPrepare + " WHERE " + pcFilter.
QUERY qCustomer:QUERY-PREPARE (cPrepare).
DATASET dsCustomer:FILL().
mhttCustQuery:QUERY-OPEN ().
```

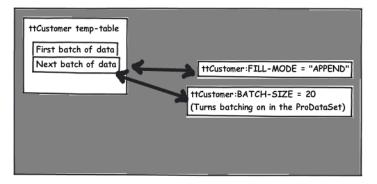
I need to duplicate those last two statements and add the additional support that batching requires. If the new input parameter signals that batching is enabled, then I need to set the **FILL-MODE** property of the DataSet's temp-table buffer. If the **FILL** going to be loading one set of rows after another into the DataSet, then it needs to append each new batch onto the end of what's already there. That's what the **AFPEND** value tells it to do:



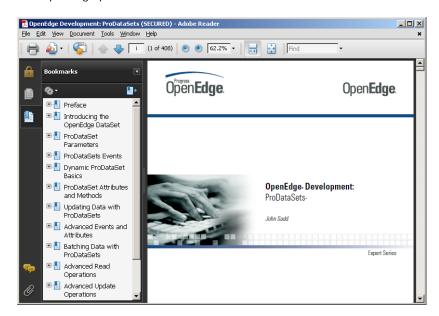
Next the code needs to signal to the DataSet that it's going to be retrieving data in batches, by setting the temp-table buffer's **BATCH-SIZE** property.



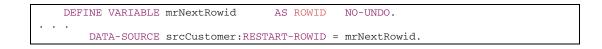
It's reasonable that the Model should determine what the right batch size is, so for simplicity's sake it's just defined in **FetchData** as a local variable **iBatchSize** with an initial value. This diagram illustrates the changes so far:



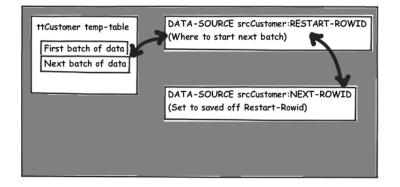
Finally, the DataSet needs to keep track of where it left off after the end of the previous batch when the caller asks it to retrieve the next one. It isn't possible to go into into all the details here, but you can learn more in the chapter on batching data in the book *OpenEdge Development: ProDataSets*, available on PSDN as part of the OpenEdge product documentation:



Basically, with each batch, the DataSet returns the **ROWID** of what will be the first row to retrieve in the next batch of data, and the new code in **FetchData** keeps track of this in the **ROWID** variable **mrNextRowid**. This is a data member of the class, that is, a variable defined in the main block of the class outside of any method, so that the value persists between calls. FetchData uses that to set the **RESTART-ROWID** property of the Customer **DATA-SOURCE** each time through:



Then **FetchData** goes ahead with the **FILL**. The **RESTART-ROWID** tells the DataSet where in the query to start filling rows. Initially **mrNextRowid** is the unknown value, so the **FILL** starts at the beginning of the database data.



The **BATCH-SIZE** set earlier in the code tells the DataSet how many rows to fill before stopping. If the **FILL** gets all the way to the end of the requested data, then the DataSet sets the **LAST-BATCH** property on the buffer, so the code checks for that, and set the new **plLastBatch** output parameter accordingly. The DataSet also sets a property in the **DATA-SOURCE** called **NEXT-ROWID** that indicates where the next batch will start, if there is one, so **FetchData** sets **mrNextRowid** data member to that value so that the next time around I can use it to set **RESTART-ROWID**:

```
DATASET dsCustomer:FILL().

plLastBatch = BUFFER ttCustomer:LAST-BATCH.

mrNextRowid = DATA-SOURCE srcCustomer:NEXT-ROWID.
```

Then the code re-opens the temp-table query on all the data that's been retrieved so far.

mhttCustQuery:QUERY-OPEN ().

The final value that needs to be calculated is the number of rows that were just added to the result set, which is the other new output parameter **piRowsAdded**. Here's the statement that calculates that value:

```
ASSIGN iNumResults = mhttCustQuery:NUM-RESULTS
piRowsAdded = iNumResults - miPrevNumResults
miPrevNumResults = iNumResults.
```

The local variable **iNumResults** holds the total number of rows that are now in the query. The **ASSIGN** statement then subtracts from that the number of rows that were previously in the query, held in another data member variable **miPrevNumResults** that keeps track of that number between calls, and finally saves off the total rows now in the query for the next time through. So **piRowsAdded** now holds the number of newly added rows to pass back from **FetchData**.

Here at the top of the class are the data member variables used in FetchData:

DEFINE VA	ARIABLE	mhttCustOuery	AS	HANDLE	NO-UNDO.
		~ 1			
DEFINE VA	ARIABLE	mrNextRowid	AS	ROWID	NO-UNDO.
		miPrevNumResults	λC	TNITECED	
DEFINE VA	AKIADUG	IIIIPIevNuIIIResuits	AS	TNIEGER	NO-UNDO.

All of these data members of the class, including the temp-table query handle, as variables defined in the main block, hold values that persist as long as the instance of the class is running. Data member variables are **PRIVATE** by default, but since they could be defined as **PROTECTED** or **PUBLIC**, it could be good practice to make the **PRIVATE** access mode explicit.

To summarize the work done so far, here's what **FetchData** looks like after these changes, marked in bold text:

```
METHOD PUBLIC VOID FetchData( INPUT pcFilter AS CHARACTER,
           INPUT plBatching AS LOGICAL,
           OUTPUT piRowsAdded AS INTEGER,
           OUTPUT plLastBatch AS LOGICAL ):
   DEFINE VARIABLE iBatchSize
                                    AS INTEGER NO-UNDO INIT 20.
   DEFINE VARIABLE iNumResults
                                  AS INTEGER NO-UNDO.
   DEFINE VARIABLE cPrepare AS CHARACTER NO-UNDO.
        cPrepare = "FOR EACH AutoEdge.Customer".
       IF pcFilter NE "" THEN
            cPrepare = cPrepare + " WHERE " + pcFilter.
        QUERY gCustomer:QUERY-PREPARE (cPrepare).
        IF plBatching THEN
        DO:
            BUFFER ttCustomer:FILL-MODE = "APPEND".
            BUFFER ttCustomer:BATCH-SIZE = iBatchSize.
            DATA-SOURCE srcCustomer:RESTART-ROWID = mrNextRowid.
            DATASET dsCustomer:FILL().
            plLastBatch = BUFFER ttCustomer:LAST-BATCH.
            mrNextRowid = DATA-SOURCE srcCustomer:NEXT-ROWID.
            mhttCustQuery:QUERY-OPEN ().
            ASSIGN iNumResults = mhttCustQuery:NUM-RESULTS
                   piRowsAdded = iNumResults - miPrevNumResults
                   miPrevNumResults = iNumResults.
        END.
        ELSE DO: /* not batching */
            DATASET dsCustomer:FILL().
            mhttCustQuery:QUERY-OPEN ().
        END.
END METHOD.
```

Now I go back to the event handler I created in the form for the **OffEnd** event. Basically it runs **FetchData** in the Model each time the user interface gets to the end of the current data in the query managed by the ProBindingSource. When **FetchData** returns, the event handler sets the **OffEndEventArgs' RowsAdded** property, and if the **ILastBatch** flag comes back **True** the code turns **Batching** off in the ProBindingSource, so that the **OffEnd** event will stop firing, since there's no more data to retrieve:

```
METHOD PRIVATE VOID moBSCustomer_OffEnd( INPUT sender AS System.Object,
INPUT args AS Progress.Data.OffEndEventArgs ):
DEFINE VARIABLE iRowsAdded AS INTEGER NO-UNDO.
DEFINE VARIABLE lLastBatch AS LOGICAL NO-UNDO.
moCustomerModel:FetchData("", TRUE, /* batching */
OUTPUT iRowsAdded, OUTPUT lLastBatch).
args:RowsAdded = iRowsAdded.
IF lLastBatch THEN
moBSCustomer:Batching = FALSE.
RETURN.
END METHOD.
```

The original **FetchData** reference in the class's constructor needs to be changed as well. It needs to pass in the ProBindingSource **Batching** property value, and get back the two new output parameters. The code is not really prepared to do anything with the **RowsAdded** value from here in the class constructor, because it's not in the **OffEnd** event handler and so the **OffEndEventArgs** object is not available, but it can at least check to see whether the initial batch is the only one, the code checks the **ILastBatch** value and is prepared to turn off **Batching** if that parameter is **True**.

ARIABLE iRowsAdded AS INTEGER. DEFINE VARIABLE llastBatch AS LOGICAL.
moCustomerModel:FetchData("", moBSCustomer:Batching, OUTPUT iRowsAdded, OUTPUT lLastBatch). IF lLastBatch THEN moBSCustomer:Batching = FALSE.

That should be all that's needed to enable data batching between the form and the model class. If I save and compile these changes and re-run the **CustomerUltraGrid** form, then when the form first comes up, it has retrieved the first batch of 20 rows. The only indication of this is that the thumb of the grid's scrollbar indicates that the majority of the available data is being displayed in the grid.

If I scroll down through the data, I can see reflected in the decreasing size of the scrollbar thumb that the **OffEnd** event is firing and new batches are being added to the data in the grid. When I've gotten to the end of the data I can sort the retrieved data just as I've done before:

	• 💊 •) 🖉 •)	2 - 3 - 🌣 🔶 - A					
St OpenEdge Editor							
Resources 🕮 🗖 🗖	IModel.cls) CustomerModel.cis 🛛 🖏		(n) 🕺 💽 Customert.	RraGrid.cl	k .	- 0
						Toolbox	ą
🖉 Samples 🔺	🛃 CustomerUlt	raGrid			-191. (+ CustonControls	
B 3 Referenced Assembles	EustomerUltrat					Microsoft Contro	
B-JL Procedure Libraries	Costomeroitra	4740		-			
B- 🗁 rcode		Cus	loner grid			OpenEdge Cont	
G G Updatable	CustomerFirstN.	ame CustomerLastName /	CustomerBithCountry	CustomerBithdate		OpenEdge Ultra	a Controls
CustomerModel.ds	Patrick	Alexander	Gemany	02/08/1983			
CustomerUltraGridUpd.cls	Robert	Baker	USA	04/23/1374			
🗌 Model.ds 📃	Evelyn	Bing	USA	12/10/1965			
Outine 📅 DB S 🛄 Prop 🔢 🛄	Helen	Brooks	Germany	07/31/1974	-		
tii ģ↓ 103 ♡	Lany	Brooks	USA	02/23/1972			
BSCustomer Progress Data BindingSource V	Mary	Buns	USA	09/02/1956			
roperties	Melissa	Buns	Gemany	10/25/1959	-		
	Richard	Bums	Gemany	01/05/1968			
AddingNew BindingComplete	Frank	Buma	USA	04/10/1984			
CancelCreateRow	Helen	Cater	USA	05/25/1951	-		
CreateRow	Susan	COllames	Germany	07/17/1972			
CurrentChanged	11		demany				
Current/temChange	<u> </u>		_		<u> </u>		
DataMemberChank	Next						
DataSourceChang -					_		
toBindingSource Designer	4				- F		
		n.					
urrentChanged	moBSCustome						
vent raised when the value of Current hanges.	🕒 Console 🕮 💽 I				_	\$ \$ 0	

Incidentally, there is a **MaxDataGuess** property on the ProBindingSource that you might want to try to set to reduce the jumping around of the scrollbar thumb as additional batches of data are retrieved and the size of the result set managed by the ProBindingSource therefore changes, but at the present time it is not possible to set **MaxDataGuess** when **Batching** is **True**.

In any case, the first row displayed when I sort by CustomerLastName is the right one – Patrick Alexander in this case – because I've retrieved all the data into the query that the ProBindingSource and the grid are attached to.

Let me compare this with what happens if I try to sort *before* I've retrieved all the batches of data. I just run the form again from the beginning. When the form first comes up, it has asked the Model to retrieve the first batch of data, as shown here:

OpenEdge Editor - Samples/CustomerU	traGrid.cl	s – OpenEdge A	rchitect - C:\Gui4Dot!	iet 🛛					-
le Edit Design Navigate Search Project	OpenEdg	je Run Window	v Help						
😁 • 🖻 🛆 🔟 🕲 🍫 • 🤅	. 🤬	• 🔗 • 🗄							
😭 🎳 OpenEdge Editor									
Resources II 😐	- IN	odel.cls	CustomerModel.cls	💫 CustomerUltraGrid.de	(Desgn) 🕺 💽 🗛	stomerU	traGrid.	sk	0
	2							Toolbox	
🗆 🎬 Samples	-	CustomerUltra	aGrid			5	101;	+ CustonControls	-
	🛃 Custon	nerUltraGrid				. O ×		* Microsoft Controls	-
III Procedure Libraries									-
🗄 🗁 rcode 🗷 🗁 Tree/Vew	-			omer grid		_		OpenEdge Controls	-
B 😂 Updatable	Custo Robe	merFirstName	Customed_astName	CustomedBirthCountry USA	CustomerBirthdate 07/13/1979		F	OpenEdge Ultra Control	İS.
CustomerModel.ds			Kennedy			+	E		
CustomerUltraGridUpd.cls	Peter		Wagner	Gemany	09/14/1907	+11			
IModel.cls	Alice		Washington	USA	05/19/1982	- 11			
E Outline 📅 DB 5 🔲 Prop 🖄 🦈	Patri	ok	Leviston	USA	03/19/1970				
臣 剑 昭	Eric		Myers	Germany	06/08/1950				
·	Laur	b	Torres	Germany	07/16/1984				
noBSDustomer : Progress Data BindingSource	Caro	1	Murphy	Germany	12/01/1982	+			
Properties Events	Thor	nas	Jenkins	Germany	12/29/1954	+			
AddingNew	Juste	0	Watson	USA	01/27/1983	+			
BindingComplete	Ath	1	Turier	USA	04/11/1990	+	-		
CancelEreateRow	Ana	orta	Roberts	USA	03/11/1971	+			
CurrentChanged	11		11000000	0.00	0.001111011	1			
CurrentHemChange		,		_		-			
DataEnor	N	est.							
DataMemberChang DataSourceChang	JII								
Datasourceunang									
ProBindingSource Designer	1						•		
CurrentChanged Event raised when the value of Current	1	mcBSCustomer							
changes.	5 Co	nsole 🔢 💽 Pi	roblems 🧟 Tasks			· · · · · · · · · · · · · · · · · · ·	De 🖉	@ @ 🖻 • 😁 •	0
r° 19				Writable					

If I immediately sort by CustomerLastName, the form is sorting only the data already retrieved, so the Customer displayed at the top of the list is the first Customer in the first batch, not the first of all Customers:

OpenEdge Editor - Samples/Eustomert				iet					JQ.
ile Edit Design Navigate Search Projec	t Op	ienEdge Run Windor	v Help						
😁 - 🖂 📥 🛛 🔏 📾 🛛 🏇 -	o -	💁 -] 🛷 -] 🗄							
📑 🎳 OpenEdge Editor									
C Resources 🕄 👘		e IModel.ds	CustomerModel.cls	🖏 CustomerUltraGrid ck	(Desgn) 🕺 💽 Oud	tomerUitr	NGrid.c	is .	- (
	7							Тоовок	7
🗉 💭 Samples		🛃 CustomerUltr	aGrid				미.;	CustomControls	
Referenced Assembles	۲Ľ	ustomerUltraGrid					_	Microsoft Controls	-
IL Procedure Libraries From reade						_			-
III - Control II				omer grid			-	OpenEdge Controls	_
B D Updatable		CustomerFirstName Evelyn	CustomerLastName / Bing	CustomerBithCountry USA	CustomerBithdate 12/10/1965		_	OpenEdge Ultra Controls	1
CustomerModel.ds			Brooks	USA	02/29/1972	+	_		
CustomerUltraGridUpd.ds		Lany							
Model.ds		Helen	Brooks	Germany	07/31/1974	- 1			
E Outline 📅 DB S 💷 Prop 23 🖤		Mary	Bums	USA	09/02/1956				
ter 44 mil		Nike	Сан	Germany	05/23/1956				
		Robert	Foster	Gemany	07/25/1970				
oBSCustomer : Progress.Data BindingSource		Alice	Ganit	USA	01/17/1956	-			
Properties Events		Melissa	Geler	Gemany	03/17/1949	+			
AddingNew		Anthony	Graham	USA	11/01/1982	+			
BindingComplete	-	Peter	Higgins	Gemany	12/03/1972	+ 1			
CancelCreateRow CreateRow		Margaret	Hunt	USA	08/11/1964	+			
CurrentChanged	1.		- That is	-	0011111001	1	_		
CurrentitemChange	114	·							
DataEvor		Next					_		
DataMemberDhang	111								
DataSourceChang	-111								
ProBindingSource Designer		4					F		
	-11		я.			_	_		
CurrentChanged Event raised when the value of Current	- 11	TroBSCustomer							
Livent raised when the value of Cultent changes.	114					_	_		-
		😂 Console 💠 🚺 Pi	oblems 🧭 Tasks		📕 X		n al	🖉 🖉 😁 😁 😁	-
r* 🖻				Writable					

Since I've changed the sort order in the middle of retrieving data, I've really invalidated what the grid is showing.

Let me make some additional changes to coordinate sorting and batching so that this doesn't happen. Back in the Model class, I need to add a few lines of code to handle the batching case. The new lines marked in bold add the following logic:

If the **BATCH-SIZE** property is set, then batching is enabled. In that case, unless the Model has already retrieved the **LAST-BATCH**, then **SortData** need to call **FetchData** to start retrieving data over again in the new sort order. Otherwise the batching will conflict with the change in sort order. So **SortData** passes in the sort criteria as the filter parameter, the first argument to **FetchData**:

/* If we're in the middle of batching, then ask FetchData to reopen the database query so that we retrieve batches in the right sequence. */ IF (BUFFER ttCustomer:BATCH-SIZE NE 0) AND (NOT BUFFER ttCustomer:LAST-BATCH) THEN FetchData(cSortString, TRUE /* Still batching */, OUTPUT iRowsAdded, OUTPUT lLastBatch). ELSE DO: mhttCustQuery:QUERY-PREPARE ("PRESELECT EACH ttCustomer " + cSortString). mhttCustQuery:QUERY-OPEN (). END.

Next, FetchData must be prepared to be called from SortData. The new condition to test for is this:

If the Model is not batching (**plBatching**); or it hasn't retrieved any batches yet, which **miPrevNumResults** tells us; or there's any filtering criteria (**pcFilter**) -- which now includes a **BY** clause being passed in from **SortData** -- then FetchData executes the code to (re-)prepare the database query.

Beyond that, because **FetchData** may now be run several times with different sort criteria, the method must empty any data from the DataSet, and reset the variables associated with batching.

IF (NO	<pre>plBatching) OR (miPrevNumResults = 0) OR (pcFilter NE "") THEN</pre>
	DO:
	<pre>cPrepare = "FOR EACH AutoEdge.Customer". IF pcFilter NE "" THEN cPrepare = cPrepare + " WHERE " + pcFilter. QUERY qCustomer:QUERY-PREPARE (cPrepare).</pre>
	DATASET dsCustomer: EMPTY-DATASET ().
	ASSIGN mrNextRowid = ? plLastBatch = FALSE miPrevNumResults = 0.
	END.

Running the form again with the sort changes, I can re-sort the data by CustomerLastName after retrieving just the first batch, and the correct first Customer now appears at the head of the list because the database guery has been restarted to re-retrieve the first batch in the new sort order:

e Edit Design Navigate Search Project				*				
CopenEdge Editor								
Resources 👷 🔍 🗆		IModel.cls	merModel.cls		an) 🔀 💽 CustomerLifts	caffeid d		0
	٣	- Coo		uscomercia da nasca (pes		:	Toolbox	
1 🖽 Samples	11	👯 CustomerUltraGrid						
Generation Generatio	11						CustomControls	_
JL Procedure Libraries	1	CustomerUltraGrid	~	atawa wid			Microsoft Controls	-
🖲 🧀 rcode		customeroitragnu					OpenEdge Controls	
TreeView			Custo	mer grid			OpenEdge Ultra Contr	ols
Updatable OustomerModel.ds		CustomerFirstName	CustomerLaptName 4	CustomerBirthCountry	CustomerBitthdate	- 1		
CustomerModel.ds		Patrick	Alexander	Germany	02/08/1983	-		
i IModel.cls		Robert	Baker	USA	04/29/1974			
Outline 📴 DB S 🛄 Prop 😫 🙄 🗖	il E	Evelyn	Bing	USA	12/10/1965			
	1 1	Helen	Brooks	Germany	07/31/1974	- 1		
rt: X + 162		Lany	Brooks	USA	02/29/1972			
oBSCustomer : Progress.Data.BindingSource 👱		Frank	Bums	USA	04/10/1984			
Properties Events		Mary	Buns	USA	09/02/1956			
AddingNew		Melicia	Buns	Germany	10/25/1959			
BindingComplete		Bichard	Burns	Germany	01/05/1968			
CancelCreateRow		Helen	Cater	USA	05/25/1951			
CurrentChanged			CCBames		07/17/1972			
CurrentitemChange		Susan	ULBarnes	Germany	0//1//19/2	•		
DataError	ЦЦ	4			<u> </u>			
DataMemberChang		Next						
DataSourceChange						_		
ProBindingSource Designer	1					F		
CurrentChanged event raised when the value of Current homores.	°	TroBSCustomer						

If I scroll down, I continue to retrieve new batches in the correct sort order.

In conclusion, in this session I showed you the **Batching** property of the ProBindingSource, which causes the binding source **OffEnd** event to fire when the end of data is detected. You can use this property and event to coordinate with a ProDataSet in your data management procedures or classes to retrieve data in batches and make the additional data available successively to your user interface.

There are more issues and techniques concerning batching than it was possible to cover in this one presentation, and for more detailed information and examples you can check out Haavard Danielsen's *Developers Corner* presentation on *Batching, Sorting, and Filtering* available on PSDN:

