

## Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.adonet/2005-09/msg00759.html>

---

- *From:* "Sorin Dolha" <[sdolha@xxxxxxxxxxxxxxxxxxxx](mailto:sdolha@xxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Sun, 18 Sep 2005 12:06:02 -0700
- 

Hello, and thank you very much for your time to provide advice.

I realize now that I didn't describe well how the client application is intended to run and what are our requirements on that part.

Well, the end-user that runs the client will need to run the application directly from a Web site (under a Security Context) as a hosted Windows Forms user control that is hosted in a web page (the user is required to have .NET Framework installed on the client machine, but not any SQL Server). That being said, I am not able to use a client side edition of SQL Server and do synchronization that way.

The client uses Isolated Storage to maintain data locally between subsequent runs of the application on the client, and we use DataSet write and read operations from XML files stored in Isolated Storage. We like this benefit of the DataSet.

Also, it is important that DataSets are features directly exposed by .NET Framework, without needing us to provide any third party components (libraries) to be downloaded on the client with the Windows User Control-based application.

And finally, we really need a relational way to store data as we have about 20 tables in different relations with each other in the database, and we need to have the same structure on the client side. The client uses the local cache typed DataSet as if it were a database (with relations defined within the DataSet), using the generated methods for each table. So we cannot use collections or other types of objects (or we can use such objects, but we would need to develop them and it's no need for them because the exact thing is done by generating the typed DataSet).

Hopefully, I cleared up the ideas about our application. As a review, I will say that the application must only use .NET Framework (no third-party components), should leverage all DataSet functions on the client (i.e. caching data between runs in the local Isolated Storage and getting and updating relational data using autogenerated methods in the DataTables of the typed DataSet), and the data needs to be automatically synchronized with a

## Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)

central server (SQL Server) in background. Of course, depending on the user who logged in to the client, only some data (a horizontal slice of the database – so all tables would be represented, but not all data in each of them) will be synchronized from the server to the client (based on the permissions of that user).

—

Sorin Dolha, DlhSoft  
MCAD, MCSD .NET

"W.G. Ryan MVP" wrote:

> Sorin:

>

> The following recommendation is based solely on what I understand to be the  
> nature of your program and not generally what I'd advise. The first thing  
> that comes to mind is using Sql Server CE or Sql Mobile , or Oracle Lite and  
> just use it as the back end – then do the synchronization there. There are  
> some killer benefits in the 2.0 framework to this approach but as you  
> mentioned, you want to stick with 1.1. What you get by doing this is a  
> trusted, tested, performant and flexible way to solve your problem.  
> Moreover, if the way synchronization needs to be handled changes in the  
> future, changing the synchronization model is pretty simple and can be done  
> almost transparently. If you roll your own (which may for reasons I'm not  
> aware of yet not be feasible) you are essentially re-creating a very very  
> complex wheel. By using Sql CE or Sql Mobile etc, you also have tremendous  
> advantages in terms of security and conflict resolution that would be next  
> to impossible (and definitely something undesirable) to recreate).

>

> I'm not sure if it's possible, but from the sounds of it, you can use other  
> events, like Button clicks or other UI Cues to do the same thing you are  
> trying to accomplish with the datatable events. Would I normally suggest  
> using the UI instead of the objects themselves. NO. But your case is  
> specific and depending on what you need, it may be a viable solution.

>

> Finally, and remember, I only say this b/c of the specific nature of your  
> app, if none of the above will work, is it possible that you can not use a  
> DataSet/DataTable as your primary object for storage? For instance, if you  
> don't need Sorting, Selecting, Computing etc and you're just using the  
> objects as data stores, then you could just roll your own collection of  
> collections and add those events in yourself. Normally (as you can tell  
> from my first suggestion), I'd not recommend rolling your own object when  
> there's already something peer reviewed and widely used (like DataSets) that  
> are already coded. But, in cases where you don't need the specific  
> functionality of a datatable/dataview/dataset, and you're just using them to  
> store data, then it may make sense. Remember that you can implement the  
> same interfaces for instances, so that you can still bind to them or  
> effectively make your new object have all the features that users would  
> normally want (I know, this sounds contradictory but what I mean is that you  
> can implement the thing partially) recreating only the features that the

Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)

> application needs.  
>  
> I know i covered vastly different approaches here and I may well have not  
> understood your requirements correctly – if that's the case, please let me  
> know and I'll take a stab at it again,  
>  
> Thanks,  
>  
> Bill  
>  
>  
>  
>  
>  
> "Sorin Dolha" <sdolha@xxxxxxxxxxxxxxxxxxxx> wrote in message  
> [news:162C393C-A419-4FA2-B38B-5EC57D06CA82@xxxxxxxxxxxxxxxxxxxx](mailto:news:162C393C-A419-4FA2-B38B-5EC57D06CA82@xxxxxxxxxxxxxxxxxxxx)  
>> Hello,  
>>  
>> I have a pretty difficult question about a hard-to-explain issue (I hope  
>> my  
>> English would be good enough to make you understand). The environment  
>> beside  
>> the problem itself may also be important, so I'm gonna explain a little  
>> the  
>> whole idea and then go back to the issue and my question. Thanks in  
>> advance  
>> for your time.  
>>  
>> I am developing a .NET 1.1-based application that uses a main central  
>> DataSet instance as a local cache storage for the current user-based slice  
>> of  
>> data stored in an SQL Server database. The application consists of several  
>> threads that may access the DataSet either to retrieve data or for  
>> updating  
>> data.  
>>  
>> A special thread in the application is set to listen to all DataSet's  
>> tables' RowChanged and RowDeleted events through a single event handler  
>> that  
>> does something like this:  
>>  
>> public void RowChanged(object sender, DataRowEventArgs e)  
>> {  
>> lock (myDataSet)  
>> {  
>> DataSet changes = myDataSet.GetChanges();  
>> if (changes != null)  
>> {  
>> myDataSet.AcceptChanges();  
>> //... Store changes in a changes queue, for later use in background  
>> synchronization to the SQL Server database (using multiple

## Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)

>> DataAdapter.Update() commands, one for each DataTable in the DataSet)  
>> }  
>> }  
>> }  
>>  
>> All the other threads that use the DataSet are using a lock(myDataSet)  
>> block  
>> in order to maintain thread safety of the operations.  
>>  
>> The idea here is to get all the changes that are done to the local DataSet  
>> in a one-by-one way, so that I can update them back to the SQL Server  
>> (like a  
>> "replication" synchronization) in a background thread and do not affect  
>> the  
>> user's experience in that process.  
>>  
>> Some would argue about the comment above that it won't be necessary to get  
>> the changes one-by-one as I could from time to time execute a  
>> myDataSet.GetChanges() in the background thread itself and update all the  
>> changes back to the database at once -- however, this is not true because  
>> I  
>> have circular relationships between the database tables, and the order of  
>> the  
>> DataAdapter.Update() would be dependent on the actual changes done in the  
>> DataSet (in order to not get exceptions regarding relationship  
>> constraints).  
>> In some cases, for example for self-referring tables, there is no way to  
>> update multiple circular changes using one single DataAdapter.Update() call  
>> (because the data is updated sequentially by the adapter, and the order of  
>> the changes is not the order of change occurrence, but in the primary key  
>> order – and in some cases, entities updated first would refer entities  
>> that  
>> would be updated later and the database would complain after the first  
>> update  
>> that the referential integrity is lost).  
>>  
>> Ok. Now back to my point. The idea above works in theory, and it also  
>> works  
>> in practice but only for changes of these DataRowAction type: Change and  
>> Delete. It doesn't work for Add. Specifically, when a new row is added in  
>> a  
>> table in the DataSet, the new row is not got by the GetChanges() call in  
>> the  
>> RowChanged event handler, although modified rows and deleted rows are. As  
>> I  
>> read on other blogs and posts, it seems that this is a bug in the DataSet  
>> of  
>> ADO .NET 1.1, and seems to be solved in .NET 2.0 (tested it myself).  
>>  
>> However, because I need to stick with 1.1 for this application, what are  
>> your suggestions regarding this issue? Specifically, how to get the Add

Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)

Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)

>> changes also? I even tried to add them manually, in case that e.RowAction  
>> ==  
>> Add, by creating a new DataSet using myDataSet.clone() and adding the  
>> e.Row  
>> in the table with the same name of the DataSet (also needed to set  
>> EnforceConstraints to false for the changes to be able to do that) instead  
>> of  
>> GetChanges() call, but I feel that this workaround is not my best choice.  
>>  
>> Also, it's out of discussion to create a new thread that would be started  
>> by  
>> the RowChanged event handler (to allow .NET add the row in the meantime,  
>> and  
>> to be gettable with GetChanges() after that), and to do the GetChanges()  
>> there – because of the problem that I cannot lock the DataSet from the  
>> event  
>> handler function and release it in another thread... And what if .NET  
>> doesn't  
>> add the row until the thread gets to the GetChanges() call...  
>>  
>> Waiting for the Commit DataRowAction is also unaproprate because the  
>> threads that add rows to the DataSet's tables do not call AcceptChanges()  
>> for  
>> each Add operation they make...  
>>  
>> I hope that there is another good and nice workaround for this problem –  
>> maybe somebody else found it before me and can share the knowledge. Or at  
>> least, advise me what other approach to take in order to reach my goals  
>> for  
>> this application.  
>>  
>> If in the meantime I will find any good workaround myself, I will post it  
>> here.  
>>  
>> Thank you again, in advance.  
>>  
>> --  
>> Sorin Dolha, DlhSoft  
>> MCAD, MCSA .NET  
>  
>  
>  
>  
>

---

• *Follow-Ups:*

◆ [Re: DataSet.GetChanges\(\) in RowChanged\(DataRowAction.Add\)](#)

◇ From: William \ (Bill) Vaughn

• *References:*

Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)

◆ **Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)**

◇ From: W.G. Ryan MVP

- Prev by Date: **Re: Passing NULL values to SqlCommand.Parameters.AddWithValue**
- Next by Date: **Re: How can I sort a datetime column in a dataset.**
- Previous by thread: **Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)**
- Next by thread: **Re: DataSet.GetChanges() in RowChanged(DataRowAction.Add)**
- Index(es):
  - ◆ **Date**
  - ◆ **Thread**