



ExpressMemData

Manual

Copyright (c) 1998-1999 Developer Express Inc.
ALL RIGHTS RESERVED

Contents

Overview	3
Getting started	4
TdxMemData	7
TdxMemData Properties	7
TdxMemData Methods	10
TdxMemData Events	14
TdxMemField	15
TdxMemField Properties	15
TdxMemFieldRec	16
TdxMemFieldRec Methods	17
TdxMemFields	17
TdxMemFields Properties	17
TdxMemFields Methods	18
Common Example	18
Index	21

Overview

The ExpressMemData component is the best way to build data awareness into your application without distributing a single dll or database engine.

You can now work directly in memory and modify small amounts of data with lightening speed. No database files, no large DLLs ... but all field types (Yes, even BLOB fields) you wish to work with. TdxMemData supports calculated and lookup fields; sorting by any field, including calculated and lookup fields; bookmarks (multiple select for grid controls); locate and lookup methods.

Getting started

You can use TdxMemData as you would any another DataSet.

Let's say we wish to create an application, and in one of its modules a user can edit (edit, delete, add) the names of Internet friends.

Data Specifications:

Object countries (Id, country name) and object Internet friends (Id, first name, last name, nickname, e-mail, birthday, interests and country).

We could use standard BDE tables, but for this example, assume we don't wish to use any database files, and of course, we don't want to distribute our app with powerful, but very large DLL's.

In this instance, we have two options:

Use only non-data aware controls.

Use TdxMemData.

The first solution is a good one if you have a lot of time or are not concerned with data validation, such as required values or correct values. Obviously, you have to do an incredible amount of work to implement controls such as lookups, etc.

Let's now show you how easy it is to use TdxMemData to resolve just this type of a problem.

1.Drop two TdxMemData components onto a form. Name the first mdCountries, the second mdIntFriends

2.Double click on the mdCountries component to call the MemData Field Editor screen and press the add button. The New Field form should now be active:

To add a new field you must fill:

Field Name

Type

Size (for string fields only)

Field Type (Default data)

Lookup definition for lookup fields only

3. Create two field ID (Integer) and Name (String, char(50)).

4. Double click on the mdIntFriends component and add its fields:

ID (Integer), FirstName (String char(25)), LastName (String char(25)), NickName (String char(10)), Email (String char(80)), birthday ((Date), interests (String char(250)) and CountryID (Integer).

5. Now we have to add the lookup field to bind mdIntFreinds.CountryID <β> mdCountries.ID

Name it CountryName, type ftString, Size 50, field type – lookup and lookup definition:

Key Field – CountryID, DataSet - mdCountries, Lookup Field - ID and Result Field – Name.

Now we have a lookup field. In both the Borland DBGrid and ExpressGrid™ by Developer Express Inc., this will be represented as a lookup combo (in the ExpressGrid, it is represented as an incremental lookup combo).

6. Now we have to set correct ID field values for our TdxMemData components. Write the following event handler for onAfterInsert:

```
procedure TForm1.dmCountriesAfterInsert(DataSet: TDataSet);  
begin
```

```

    DataSet.FindField('ID').AsInteger :=
        DataSet.FindField('recid').AsInteger;
end;

```

You can use this code for both components: dmCountries and dmIntFriends. 'RECID' is the field that is created for each TdxMemData. You can see it at design time in the Object Inspector or TdxMemData fields editor.

It is a unique, Auto Incremental field. In this demo, we use it as an index field to get the next value for our ID fields. It can be used as KeyField in the dxDBGrid control.

7. Now we have to write code to save and restore data in a text file. Here it is:

```

//Load the data on form OnCreate event, check if file exists
procedure TForm1.FormCreate(Sender: TObject);
var
    FileName : String;
begin
    FileName := ExtractFileDir(Application.ExeName) + '\countries.txt';
    if FileExists(FileName) then
        mdCountries.LoadFromTextFile(FileName);
    FileName := ExtractFileDir(Application.ExeName) + '\intfriends.txt';
    if FileExists(FileName) then
        mdIntFriends.LoadFromTextFile(FileName);
end;

//Save the data on form destroy event

procedure TForm1.FormDestroy(Sender: TObject);
begin
    mdCountries.SaveToTextFile(
        ExtractFileDir(Application.ExeName) + '\countries.txt');
    mdIntFriends.SaveToTextFile(
        ExtractFileDir(Application.ExeName) + '\intfriends.txt');
end;

```

8. Now drop two DataSources and name them dsCountries and dsIntFriends then link them with your dxMemData components.

9. Set Active to True for both TdxMemData components. The final step is to link your DataSources with individual data controls.

And that's all you need to do!

TdxMemData

TdxMemData is directly inherited from the TDataSet component.

Unit

dxMemData

```
TdxMemData = class(TDataSet);
```

Description

TdxMemData works directly with memory and allows you to work with small amounts of data with lightening fast speeds.

TdxMemData supports:

- Calculated and lookup fields
- Sorting by any field, including calculated and lookup fields
- Bookmarks (multiple select for grid controls)
- Locate, lookup methods

Note: This component does not work with Delphi™ 2, and CPP Builder™ 1.

TdxMemData Properties

In TdxMemData

Data	public / read only
DelimiterChar	public / read/write
IsLoading	public / read/write
ReclIdField	public / read only
ReadOnly	published / read/write
SortedField	published / read/write
SortOptions	published / read/write

Derived from TDataSet

Active	published / read/write
Filter	published / read/write

Data property

read only

Returns MemData's fields definitions.

```
property Data : TdxMemFields;
```

Description

Use the Data property to get access to the MemData's fields definitions.

DelimiterChar property**read/write**

Defines the separator char.

```
property DelimiterChar : Char;
```

Description

Use the DelimiterChar property to define the column separator character which will be used by MemData when you store or restore data within MemData using the SaveToTextFile and LoadFromTextFile methods.

Example (Delphi)

```
//Save data to a text file with commas as column separators

procedure TForm1.Button1Click(Sender: TObject);
begin
    //Define commas as column separators
    dxMemData1.DelimiterChar := ',';
    dxMemData1.SaveToTextFile(ExtractFileDir(Application.ExeName) +
                              'mdata.txt');
end;

//Load data from a text file with commas as column separators

procedure TForm1.Button2Click (Sender: TObject);
begin
    with dxMemData1 do begin
        DisableControls;
        DelimiterChar := ',';
        LoadFromTextFile(ExtractFileDir(Application.ExeName) +
                          'mdata.txt');

        EnableControls;
    end;
end;
```

See also

TdxMemData.LoadFromTextFile, TdxMemData.SaveToTextFile

IsLoading property**read/write**

Indicates whether the data loading process of MemData is active.

```
property IsLoading : Boolean;
```

Description

Use this property if you wish to know whether MemData is loading data. You can also employ this property when using your own methods of data loading.

RecIdField property**read only**

Returns a pointer to the 'RecId' field of MemData

```
property RecIdField : TField;
```

Description

Use the RecldField property to return a pointer to the 'Recld' field of the MemData. 'Recld' is invisible and is created automatically during the creation of a TdxMemData component. The RecldField is a unique field.

Note: If you do not have a unique field and want to use the grouping capabilities of the ExpressGrid™, then use this field as the KeyField property of the dxDBGrid.

Example (Delphi)

```
// Set the unique field after insert operation.
procedure TForm1.dxDMemData1AfterInsert(DataSet: TDataSet);
begin
    DataSet.FindField('id').AsInteger :=
        DataSet.FindField('recid').AsInteger;
end;
```

ReadOnly property

read/write

Specifies whether MemData is read-only.

```
property ReadOnly : Boolean;
```

Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in MemData. By default, ReadOnly is False, therefore users can potentially alter data.

SortedField property

read/write

Determines the field name by which MemData is sorted.

```
property SortedField : String;
```

Description

Use SortedField to specify the field name by which Memdata is sorted.

Note: By default, MemData will sort the data of this field in ascending order. To change sorting order you have to use the SortOptions property. Also, using the SortOptions property you can set case insensitive sorting.

See also

TdxMemData.SortOptions

SortOptions property

read/write

Specifies the sort type of MemData.

```
type TdxSortOption = (soDesc, soCaseInsensitive);
    TdxSortOptions = set of TdxSortOption;
property SortOptions : TdxSortOptions;
```

Description

Use SortOptions to specify the sort type of MemData.

The following values are available:

Value	Meaning
soDesc	If soDesc is active, MemData sorts in descending order, otherwise it sorts in ascending order.

`soCaseInsensitive` Determines whether MemData sorts using case insensitive mode.

By default, all of these options are inactive.

See also

TdxMemData.SortedField

TdxMemData Methods

<code>FillBookMarks</code>	<code>public</code>
<code>GetRecNoByFieldValue</code>	<code>public</code>
<code>LoadFromBinaryFile</code>	<code>public</code>
<code>LoadFromStream</code>	<code>public</code>
<code>LoadFromTextFile</code>	<code>public</code>
<code>MoveCurRecordTo</code>	<code>public</code>
<code>SaveToBinaryFile</code>	<code>public</code>
<code>SaveToStream</code>	<code>public</code>
<code>SaveToTextFile</code>	<code>public</code>
<code>SupportedFieldType</code>	<code>public</code>
<code>UpdateFilters</code>	<code>public</code>

FillBookMarks method

Fills MemData bookmarks.

```
procedure FillBookMarks;
```

Description

Call this method only when you insert data using direct access without standard DataSet methods like Append or Insert.

GetRecNoByFieldValue method

`GetRecNoByFieldValue` returns a record number by a particular field value.

```
function GetRecNoByFieldValue(Value : Variant; FieldName:String):
    Integer; virtual;
```

Description

Call the `GetRecNoByFieldValue` to obtain a particular `RecNo`. Use the `Value` and `FieldName` parameter to specify required `RecNo`. If an applicable record is not found, the `GetRecNoByFieldValue` returns -1. If there are more than one record with a particular value, the `RecNo` of the first record is returned.

LoadFromBinaryFile method

Loads data from a binary file.

```
procedure LoadFromBinaryFile(FileName : String);
```

Description

Use this method to load data from a binary file specified by the FileName parameter.

Note: A file specified by the FileName must contain data in MemData internal format (For example, data written to a file by another MemData using the SaveToBinaryFile method).

Example (Delphi)

```
//Load data from a binary file
procedure TForm1.Button1Click (Sender: TObject);
begin
    with dxMemData1 do
    begin
        DisableControls;
        LoadFromBinaryFile(ExtractFileDir(Application.ExeName) +
                           'mdata.bin');
        EnableControls;
    end;
end;
```

See also

TdxMemData.LoadFromStream, TdxMemData.LoadFromTextFile, TdxMemData.SaveToBinaryFile, TdxMemData.SaveToStream, TdxMemData.SaveToTextFile

LoadFromStream method

Loads data from the stream.

```
procedure LoadFromStream(Stream : TStream);
```

Description

Call LoadFromStream to populate MemData with data from the stream. The Stream parameter specifies the name of the stream from which to read data.

Note: Stream must contain data in MemData internal format (For example, data written to a stream by another MemData using the SaveToStream method).

Example (Delphi)

```
//Copying data from one MemData component to another
procedure TForm1.Button1Click(Sender: TObject);
var
    fMem : TMemoryStream;
begin
    fMem := TMemoryStream.Create;
    //Store 1st MemData data to the stream
    dxMemData1.SaveToStream(fMem);
    //Set the Stream position
    fMem.Position := 0;
    //Populate the 2nd MemData data from the stream,
    //containing data from the 1st MemData
    dxMemData2.LoadFromStream(fMem);
    fMem.Free;
end;
```

See also

TdxMemData.LoadFromBinaryFile, TdxMemData.LoadFromTextFile, TdxMemData.SaveToBinaryFile, TdxMemData.SaveToStream, TdxMemData.SaveToTextFile

LoadFromTextFile method

Populates data from a text file.

```
procedure LoadFromTextFile(FileName : String); dynamic;
```

Description

Use this method to load data from a text file specified by the FileName parameter. By default, when loading data from a text file 'TABs' will be used as column separators. If you have a text file with comma as a separating character, you must set the DelimiterChar property first.

Note: When you use the LoadFromTextFile method, you are not guaranteed that all data will be loaded properly. If regional settings are different from original settings with which the file was created, loading errors may occur. If you want to be certain that your file will be read properly, you have to use the SaveToBinaryFile and LoadFromBinaryFile methods.

Example (Delphi)

```
//Load data from a text file with commas as column separators
```

```
procedure TForm1.Button2Click (Sender: TObject);  
begin  
    with dxMemData1 do begin  
        DisableControls;  
        DelimiterChar := ',';  
        LoadFromTextFile(ExtractFileDir(Application.ExeName) +  
                        'mdata.txt');  
        EnableControls;  
    end;  
end;
```

See also

TdxMemData.DelimiterChar, TdxMemData.LoadFromBinaryFile, TdxMemData.LoadFromStream, TdxMemData.SaveToBinaryFile, TdxMemData.SaveToStream, TdxMemData.SaveToTextFile

MoveCurRecordTo method

Moves the current record to a new position within MemData.

```
procedure MoveCurRecordTo(Index : Integer);
```

Description

Use this method if you wish to move the current active record to another position specified by the Index parameter.

Example (Delphi)

```
//Move specified record to the start of the MemData  
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    dxMemData1.DisableControls;  
    //Move record  
    dxMemData1.MoveCurRecordTo(0);  
    dxMemData1.EnableControls;  
    //Go to the first record  
    dxMemData1.First;  
end;
```

SaveToBinaryFile method

Saves data to a binary file.

```
procedure SaveToBinaryFile(FileName : String);
```

Description

Use this method if you want to save data to a binary file, specified by the FileName parameter.

Example (Delphi)

```
//Save data to a binary file
procedure TForm1.Button1Click(Sender: TObject);
begin
    dxMemData1.SaveToBinaryFile(ExtractFileDir(Application.ExeName) +
                               'mdata.bin');
end;
```

See also

*TdxMemData.LoadFromBinaryFile, TdxMemData.LoadFromStream,
TdxMemData.LoadFromTextFile, TdxMemData.SaveToStream, TdxMemData.SaveToTextFile*

SaveToStream method

Saves data to the stream.

```
procedure SaveToStream(Stream : TStream);
```

Description

Call SaveToStream to write a MemData's data to the stream. The Stream parameter specifies the name of an existing stream.

See also

*TdxMemData.LoadFromBinaryFile, TdxMemData.LoadFromStream,
TdxMemData.LoadFromTextFile, TdxMemData.SaveToBinaryFile, TdxMemData.SaveToTextFile*

SaveToTextFile method

Saves data to a text file.

```
procedure SaveToTextFile(FileName : String); dynamic;
```

Description

Use this method if you wish to save data to a text file, specified by the FileName parameter. By default, when saving data to a text file 'TAB's will be used as column separators. If you want to use a different column separator, you have to use the DelimiterChar property.

Note: When you use the SaveToTextFile method, you are not guaranteed that later all data will be loaded properly. If regional settings are different from original settings, with which the file was created, loading errors may occur. To be certain that your file will be read properly regardless of regional settings, you have to use the SaveToBinaryFile and LoadFromBinaryFile methods.

Example (Delphi)

```
//Save data to a text file with commas as column separators
procedure TForm1.Button1Click(Sender: TObject);
begin
    //Define commas as column separators
```

```

dxMemData1.DelimiterChar := ',';
dxMemData1.SaveToTextFile(ExtractFileDir(Application.ExeName) +
                          'mdata.txt');
end;

```

See also

TdxMemData.DelimiterChar, TdxMemData.LoadFromBinaryFile, TdxMemData.LoadFromStream, TdxMemData.LoadFromTextFile, TdxMemData.SaveToBinaryFile, TdxMemData.SaveToStream

SupportedFieldType method

Determines whether a particular FieldType is supported by TdxMemData.

```
function SupportedFieldType(AType: TFieldType): Boolean; virtual;
```

Description

Use the SupportedFieldType to determinate whether a field type is supported by MemData (As determined by the AType parameter). If a given field type is supported, the SupportedFieldType returns True; Otherwise – False.

Note: With the release of v1.4, BLOB fields are supported.

UpdateFilters method

Updates MemData's filters.

```
procedure UpdateFilters; virtual;
```

Description

Call this method to update MemData's filters which you can define in the OnFilterRecord event. Use this method when you make changes to filter definitions.

TdxMemData Events**Derived from TDataSet**

AfterCancel	published
AfterClose	published
AfterDelete	published
AfterEdit	published
AfterInsert	published
AfterOpen	published
AfterPost	published
AfterScroll	published
BeforeCancel	published
BeforeClose	published
BeforeDelete	published
BeforeEdit	published
BeforeInsert	published
BeforeOpen	published

BeforePost	published
BeforeScroll	published
OnCalcFields	published
OnDeleteError	published
OnEditError	published
OnFilterRecord	published
OnPostError	published

TdxMemField

TdxMemField is the object used by a MemData component to store particular memory field definitions.

Unit

dxMemData

```
TdxMemField = class(TCollectionItem);
```

Description

The TdxMemField object allows you to obtain access to the memory field definition. Each of MemData's memory fields is a list of pointers to field values.

TdxMemField Properties

DataSet	public / read only
Field	public / read only
HasValues	public / read/write
MemFields	protected / read only
ValueList	public / read only
Values	public / read only

DataSet property

read only

Specifies a corresponding dataset.

```
property DataSet : TdxMemData;
```

Description

Use this property to get access to a corresponding MemData component.

Field property

read only

Specifies a corresponding field.

```
property Field : TField;
```

Description

Use this property to get access to a corresponding field definition.

HasValues property**read/write**

property HasValues[Index : Integer] : Byte;

Description

This property is used internally by SaveToStream and Locate methods. You should never really have a need to use it directly.

MemFields property**read only**

Determines the corresponding TdxMemFields object.

property MemFields : TdxMemFields;

Description

Use the MemFields property to get access to a TdxMemFields object which contains the MemField.

ValueList property**read only**

Specifies the list, containing MemField's values.

property ValueList : TList;

Description

Use this property to get the TList object which contains MemField's values.

See also

TdxMemField.Values

Values property**read only**

Determines the values of a MemField.

property Values[Index : Integer] : Pointer;

Description

Use the Values property to read and write a particular MemField value determined by the Index parameter.

See also

TdxMemField.ValueList

TdxMemFieldRec**Unit**

dxMemData

TdxMemFieldRec = **class**(TObject);

Description

TdxMemFieldRec is used by a MemData component to create memory fields. You may use the TdxMemFieldRec object when you fill the values of MemData directly.

TdxMemFieldRec Methods

Create	public
Destroy	public

Create method

Creates a new TdxMemFieldRec.

constructor Create(Buffer : Pointer; Size : Integer);

Description

Use this method to create a new TdxMemFieldRec. The Buffer parameter determines the pointer to the buffer, containing data for a field's value. The Size parameter determines the size of the buffer.

Destroy method

Destroys a TdxMemFieldRec object.

destructor Destroy; override;

Description

This method destroys a TdxMemFieldRec instance.

TdxMemFields

TdxMemFields is a collection of TdxMemField objects used by a MemData component to store the memory fields definitions.

Unit

dxMemData

TdxMemFields = **class**(TCollection);

Description

TdxMemFields is a container for MemData's memory fields. It is a collection of lists, where each item is a TdxMemField object.

TdxMemFields Properties

DataSet	public / read only
Items	public / read only
RecordCount	public / read only

DataSet property

read only

Specifies a corresponding dataset.

property DataSet : TdxMemData;

Description

Use this property to get access to a corresponding MemData component.

Items property

read only

Lists the MemField definitions that describe each memory field in a MemData.

```
property Items[Index : Integer] : TdxMemField;
```

Description

Use Items to access a particular MemField's definition. Specify the Index parameter to access a the particular MemField definition. The Index parameter is in the range from 0 to Count - 1.

RecordCount property

read only

Indicates the total number of records in a corresponding MemData.

```
property RecordCount : Integer;
```

Description

The RecordCount property returns the total number of records in a corresponding MemData.

TdxMemFields Methods

IndexOf public

IndexOf method

Locates the MemField object in the MemFields array.

```
function IndexOf(Field : TField): TdxMemField;
```

Description

Call the IndexOf to get the MemField specified by the Field parameter. If the requested field is not found. IndexOf returns *nil*.

Common Example

This example demonstrates the use of TdxMemData.Data, TdxMemData.FillBookMarks, TdxMemField.Values, TdxMemField.ValueList, TdxMemFieldRec.Create, TdxMemFields.Items.

The example shows:

- The standard and easiest way to use the Append and Post methods of the abstract DataSet.
- The second and somewhat more difficult way to add records directly from memory.

The second variant requires a little more coding, but it is significantly faster (Our tests show a performance improvement by a factor of ten). In our examples, it takes 5 seconds to add 10,000 records using the first variant and only 0.5 seconds by using the second.

Note: In both variants, you must employ the DisableControls / EnableControls methods. If you are using the SortedField property, you must empty it and then restore it.

The first variant – Append / Post

Delphi

```

//Disable the data controls to prevent flickers
mData.DisableControls;
//Delete all records (free memory)
mData.Close;
mData.Open;

//Add 10 000 records.
for i := 0 to 9999 do
begin
    mData.Append;
    mData.FindField('firstname').AsString := 'FistName ' +
                                                IntToStr(i + 1);
    mData.FindField('lastname').AsString := 'LastName ' +
                                                IntToStr(i + 1);
    mData.Post;
end;

//Go to the first record
mData.First;

mData.EnableControls;

```

C++ Builder

```

//Disable the data controls to prevent flickers
mData->DisableControls();
//Delete all records (free memory)
mData->Close();
mData->Open();

//Add 10 000 records.
for (int i = 0; i < 10000; i++)
{
    mData->Append();
    mData->FindField("firstname")->AsString = "FirstName " +
                                                IntToStr(i + 1);
    mData->FindField("lastname")->AsString = "LarstName " +
                                                IntToStr(i + 1);
    mData->Post();
}

//Go to the first record
mData->First();
//Enabled the data controls
mData->EnableControls();

```

The second variant – Direct access to memory

Delphi

```

//Disable the data controls to prevent flickers
mData.DisableControls;
//Delete all records (free memory)
mData.Close;
mData.Open;

pch := StrAlloc(255);

```

```
//Add 10 000 records.
//constructor Create(Buffer : Pointer; Size : Integer);
//The size is the field size.
//If the Buffer is not NULL then the data will be copied
//In all other cases the field value will be NULL (filled by zeros).
for i := 1 to 10000 do
begin
    //Assign value to internal RecId field. It should start from 1
    mData.Data.Items[0].ValueList.Add(TdxMemFieldRec.Create(@i, 4));
    //Assign value to Id field.
    mData.Data.Items[1].ValueList.Add(TdxMemFieldRec.Create(@i, 4));
    //Assign value to First name field.
    pch := StrPCopy(pch, 'First Name ' + IntToStr(i));
    mData.Data.Items[2].ValueList.Add(TdxMemFieldRec.Create(pch,
                                                             mDataFirstField.DataSize));
    //Assign value to Last name field.
    pch := StrPCopy(pch, 'Last Name ' + IntToStr(i));
    mData.Data.Items[3].ValueList.Add(TdxMemFieldRec.Create(pch,
                                                             mDataLastField.DataSize));
end;

//Fill bookmarks
mdata.FillBookmarks;

StrDispose(pch);
//Go to the first record
mData.First;

//Enabled the data controls
mData.EnableControls;
```

Index

C	
Create.....	17
D	
Data	7
DataSet	15, 17
DelimiterChar.....	8
Destroy.....	17
F	
Field.....	15
FillBookMarks	10
G	
GetRecNoByFieldValue	10
H	
HasValues	16
I	
IndexOf.....	18
IsLoading	8
Items	18
L	
LoadFromBinaryFile	10
LoadFromStream	11
LoadFromTextFile	12
M	
MemFields.....	16
MoveCurRecordTo	12
R	
ReadOnly.....	9
RecIdField	8
RecordCount.....	18
S	
SaveToBinaryFile	13
SaveToStream.....	13
SaveToTextFile.....	13
SortedField	9
SortOptions.....	9
SupportedFieldType	14
U	
UpdateFilters	14
V	
ValueList.....	16
Values	16