oksDBEngine 1.2

Home Edition / Professional Edition

Inhaltsverzeichnis

About oksDBEngine

oksDBEngine is a standalone, singlefile databaseengine with full SQL-support.

With the provided nonvisual oksDBEngine components for Borlands® Delphi© IDE Version 7 - 2006 oksDBEngine is designed for replacing Borland®'s Database Engine (BDE©). Provided components for Delphi are TOKSDBEDatabase, TOKDBETable and TOKSDBQuery. These components require Borland®'s Databasesystem (not delivered with Personal Editions of Delphi©), espacially Borland®'s TDataSource as transmitter between databaseengine's datasets (table or query) and visual components like TDBGrid. In this case oksDBEngine's database-components work with all of Borland®'s or Third Party's visual database-components.

oksDBEngine Home Edition is Freeware. That means you could use oksDBEngine for your personal use. It's not allowed to sell your applications or databases with oksDBEngine files. A reminderscreen is shown everytime a new databasesession is created. If you would like to sell your applications or databases created with oksDBEngine you have to buy the royalty free Professional Edition without any limitations.

Standalone database engine

That means oksDBEngine runs outside Borlands® Delphi© IDE without the need of Delphi© beeing installed. For this use the oksDBEngine Admintool.

Singlefile database engine

That means that oksDBEngine's tables are completely integrated in the databaseengine: you have only one file on your harddisk, that contains all tables. As a software developper you just have to provide only one file with your database applications.

Client-Server database engine

oksDBEngine is designed as a Client-Server-Databaseengine, so it can be run as a client for single user access or as client-server with multiple useraccess in a network. The multiple user access ist limitated to 99 simoultanous accesses. We're working for further versions of oksDBEngine with more simoultanous acceesses. If you're recieving our newsletter we will inform you about new implementations.

BDE-replacement

That means that oksDBEngine requires Borland®'s Databasesystem for developping databse applications indeed, but even not the BDE©.

SQL-Support

oksDBEngine supports fully the SQL-92-standard. With TOKSDBEQuery you have unlimited access to SQL.

How to order the Professional Edition

To order the oksDBEngine Professional Edition please fill out the orderform at the end of this document an send it to support@olkrosoft.de. You will recieve a order confirmation via email and our bank details. Please note we're only shipping via email and not until we have recieved the payment (that means we only accept prepayment).

Installation

1) Copy the provided zip-file that contains the package of oksDBEngine Home Edition or Professional Edition to any folder (directory) you want

2) Unzip all files contained in the zip-file to this directory

3) Run your delphi-ide and/or close all active projects or forms

4) From the file-menu select „open“ and select the oksDBEngine-package (.dpk)

5) Since Delphi 2005 (Version 9) then you have to choose the option „for Delphi 32“

6) The package with all included necessary files should be opened

7) Compile and install the package

8) In the IDE's components-tab should be now the tab „oksDBEngine“ with 3 components: TOKSDBEDatabase, TOKSDBETable and TOKSDBEQuery.

First steps

Before using the components you first should run the admintool (oksDBE_Admin.exe) wich is contained in the package-zipfile.

First of all let's create a database.

In the Mainmenu select „Datenbank(Database)“, „Erstellen(Create)“. In the Filedialog shown enter a desired filename for the database to be saved on harddisk (e.g. myFirstDatabase) and hit „ok“. The databasefile has been created, that's all. On the left panel of oksdBEngine-Admintool the main parameters of the created database are now shown in a treeview. These parameters are just for information and could not be modified except the encryption.

A new table's born

After creating the database let's create a table for our database.

In the treeview on the left side select the first node, wich has the filename of the created database. In the Mainmenu select the item „Tabelle (Table)“ and then the item „Erstellen (Create)“. A new window is shown.

- First enter the tablename for the new table in the editfield on the top. Note: this is no filename but the table's real name in the database (e.g. „Table_1“).
- Now enter the grid on the left side (called „Felder (Fields)“. Enter a fieldname in the grids column named „Name“ (e.g. Field_1).
- In the column „Type“ select a type for the field (e.g. „String“ for a stringfield).
- Now you could choose the „Size“ of the stringfield (e.g. 200).
- If the field should be required for input every time a new record is created then choose „True“ in the column „Erforderlich (Required)“.
- If the field should have a default value every time a new record is created enter a default value in the column „Vorgabe (Default)“ e.g. „Customer“.
- For numeric fields such as interger- or floatfields you could choose a minimum and maximum value for the field each time a new record is created (e.g. „10“ and „100“).

Repeat these steps as long as you would add fields to the new table.

Next enter the top right grid called „Indeces“ in the table's window for adding an Index to the table. Note: every table should have at least on index for sorting-capabilities without using SQL.

Select the column „Typ (Type)“ and enter the type of the index (PRIMARY OR UNIQUE).

Select the column „Name“ and enter a explicite name for the index (e.g. „Idx_Main“).

Next enter the bottom right grid called „Indexfelder (Indexfields)“ in the table's window for applying the index to an field of the table.

- In the column „Spalte (Column)“ select the field you wish to add the created index.
- In the column „Gross-Kleinschreibung (CaseInsensitive)“ you could enter „True“

or „False“ if sorting should regard uppercase strings.

- In the column „Aufsteigend (Ascending)“ you could enter „True“ or „False“ for ascending or descending sorting-capabilities.

Now hit the OK-Button. That's it. A new table's born and shown in the treeview of the main window.

Developpers choice

After creating the database and it's tables close the oksDBEngine-Admin and place the responding components TOKSDBEDatabase and TOKSDBETable and from the DataAccessing-tab TDataSource on your application's form. From the DataControls-Tab place the TDBGrid on the form.

Note: Creating databases and tables is also possible via sourcecode-programming with the components TOKSDBEDatabase and TOKSDBETable. Please referr to the components-description below.

Enter the TOKSDBEDatabase properties as follows:

- Set Client-Server to True
- Select the DatabaseFilename you have created with the Admintool (e.g. myFirst-Database)
- Enter a Databasename for identification (e.g. DB_Customers)
- Set ConnectedToDatabase to True

Enter the TOKSDBETable properties as follows:

- Select the Databasename (e.g. „DB_Customers“ from sample above)
- Select the Tablename (e.g. „Table_1“ from sample above)
- Select the Indexname (e.g. „Idx_Main“ from sample above)
- Set Active to True

Enter the TDataSource properties as follows:

- Select the Dataset (e.g. „OKSDBETable1“)

Enter the TDBGrids properties as follows:

- Select the property Datasource responding to the TDataSource (e.g. „Data-Source1“)
- Doubleclick the property Columns
- Add a new item (TColumn)
- Select the TColumn's property FieldName to the responding field of your table you have created in the example above (e.g. „Field_1“).
- Repeat the adding Tcolumns as long as you would have fields shown in the TDB-Grid.

That's it. Your Database-Application is about to start. As you see TOKSDBETable is used the same way as Borland's TTable. And that goes for TOKSDBEQuery too.

oksDBEngine Admin

Menu Database

Open

Open's an existing database. The fileextension for oksDBEngine is „.odb“. The item „open“ is only available if no database is opened or if current opened database is selected in treeview.

Create

Select „create“ to create a new database. After selecting this item a savedialog will appear for entering a new database-filename or overwriting an existing. If an database is already open the opened database will be closed and the new database opened instead. The item „create“ is only available if no database is opened or if current opened database is selected in treeview.

Close

Select „close“ to close an opened database. The item „close“ is only available if no database is opened or if current opened database is selected in treeview.

Delete

Select the item „delete“ to delete the opened database from harddisk. If you confirm the appearing warning dialog the database and all it's tables would be deleted and are lost. The item „delete“ is only available if no database is opened or if current opened database is selected in treeview.

Encryption by Password

Select the item „Encryption by Password“ to encrypt the opened database. A password-dialog will appear. First you have to activate the checkbox „Activate Encryption“ to make the password-fields editable. Then enter a password in the first adminpassword-editfield and confirm your password by retyping it in the second editfield. Press the ok-button to process encryption. The encryption will be done by the Twofish-128 algorithm.

The next time you open the encrypted database you have to enter the password in an apperaing password-dialog.

Important Note:

If you forgot the password there's no chance to open the database again.

Refresh

In some rare cases or in client-server mode it might be necessary to refresh the treeview for showing the modifications you or an another user just might have done similtaneous.

Menu Table

Create

Select the item „create“ to create a new table in opened database. To access this item the opened database or an another existing table in the treeview must be selected. After selecting this item a new window is shown. For filling out the necessary data such as tablename, fields and indices referr to the example above in chapter „A new table's born“.

Create by structure of selected table

Select this item to create a new table with the structure of an existing table. Select in treeview the existing table which structure should be inherited to the next table. After selecting this menuitem a new window is shown. For filling out the necessary data such as tablename, fields and indices referr to the example above in chapter „A new table's born“.

Rename

Select this item to rename the table selected in treeview. A renamedialog will be shown where you can enter the new tablename.

Empty

Select this item to empty the table selected in treeview. If you confirm the appearing warningdialog all data will be removed from the table and are lost. Note that this command doesn't affect the structure of an table but only the data in.

Delete

Select this item to delete the table selected in treeview from the current opened database. If you confirm the appearing warningdialog the table with all it's contents and data will be removed from database and is lost.

Structure

Select this item to retrieve information about the structure of the table selected in treeview. A new window will be shown, this is the same window as for creating a new table, except you could not apply any modifications.

Restructure

Select this item to modify the structure of the table selected in the treeview. A new window is shown, the same as for creating a new table.

Export

Select one of the export-menuitems to export the data from the table selected in treeview to an sql-statement-file (sql), to a comma-separated file (csv), to a richtext fileformat (rtf), to Microsoft® Word© (doc), to Microsoft® Excel© (xls), or webbased fileformat (html).

Menu SQL

oksDBEngine is full compatible to SQL-92. If you select the menuitem „SQL" an new window is shown.

On the left side is an richeditbox, where to enter the sql statements. Reserved words, brackets, strings and integervalues are highlighted with several colors and / or bold font like in delphi's codeeditor-window. Type in the desired statement and hit the executebutton. If statements are valid the results of affecting the rows in the statement will be shown on rightsided grid. Please referr to extern SQL-Helpfiles for usage of SQL, because the issue „SQL" is too large to be printed here.

Components

Please note most desriptions made in this document are directly imported from the original delphi db-helpfile, because most properties, methods and events derive from Borlands BDE. It's recommended the user is familiar with the basics of Borlands BDE according using Tables and Queries at designtime as well as at runtime. For further information on implemented classes from Borlands BDE please referr to the original help file.

TDatasource (Delphi's DataAccess)

Properties

For all properties of TDataSource please referr to Delphi's Helpfile.

Methods

For all procedures of TDataSource please referr to Delphi's Helpfile.

Events

For all events of TDataSource please referr to Delphi's Helpfile.

TOKSDBEDatabase

Properties

AdminPass

AdminPass: String; read or write

Reads or sets the password for database-encryption. Set the AdminPass in property-editor of the component or in sourcecode.

ClientServer

ClientServer: Boolean; read or write

If set to true, client-server-property will allow simoultanous access by multiple users. Otherwise only one user can access or modify the database properties or contents.

ConnectedToDatabase

ConnectedToDatabase: Boolean; read or write

If set to true database-component will be connected to the database on harddisk. Otherwise it will be disconnected. Set to treu is required for accessing and modifying the database.

DatabaseFileName

DatabaseFileName: String; read or write

Reads or writes the filename of database on harddisk. Standard fileextension is „.odb“. Before writing the databasefilename the property ConnectedToDatabase must be set to false.

DatabaseName

DatabaseName: String; read or write

Reads or writes the name of the database for explicit assignment. The databasename represents the database for accessing by TOKSDBETable and TOKSDBEQuery.

DataSetCount

DataSetCount: Integer; read only

Derived from TCustomConnection DataSetCount indicates the number of datasets associated with TOKSDBEDatabase.

DataSets

DataSets[Index: Integer]: TOKSDBEDataSet;; read only

Derived from Tdatabase provides DataSets an indexed array of all active datasets for a database component. Use DataSets to access active datasets associated with a database component. An active dataset is one that is currently open. If a ClientDataSet uses the connection component to connect to a database server, DataSets lists the internal dataset that the client dataset creates to access data, not the client dataset itself.

DisableTmpFiles

DisableTmpFiles: Boolean; read or write

Setting DisableTmpFiles to true, all modifications on database or its contents will not be stored unless unless the Post-method is explicitly called.

Exclusive

Exclusive: Boolean; read or write

Derived from Tdatabase the property Exclusive enables an application to gain sole access to a database.

Use Exclusive to prevent other applications from accessing a database while this application is using it. Before opening the database, set Exclusive to true. The default value for Exclusive is false, allowing other applications to use the opened database.

When Exclusive is true and the application successfully opens the database, no other application can access it. If the database for which the application has requested exclusive access is already in use by another application, an exception is raised. To handle such exceptions, write an exception handler. All tables in the database are effectively opened exclusively, as other applications would be prevented from opening the database and the tables would be inaccessible.

A database must be closed before changing the setting of the Exclusive property. Do not set Exclusive to true at design time if you also intend to set the ConnectedToDatabase property to true at design time. In this case an exception is raised because the database is already in use by the IDE.

Exists

Exists: Boolean; read only

If the result of Exists is true, the databasefilename exists on harddisk. Use this property at runtime before creating or replacing a database.

Handle

Handle: TOKSDBESession; read or write

Derived from TSession Handle specifies the Handle of an database-session. Use Handle only to bypass TOKSDBESession methods and write directly to the oksDBEngine implementations Derived from TSession. Many oksDBEngine function calls require a handle parameter. Handle is assigned an initial value when a session is activated.

Note: Do not use this property unless an application requires oksDBEngine functionality not available through library components.

HandleShared

HandleShared: Boolean; read or write

Derived from TDatabase HandleShared specifies whether to share a database handle. Use HandleShared to indicate that a database component can share its handle in a session component. Set HandleShared to true to avoid namespace conflicts for database components that appear in a remote data module, or that appear in data modules you inherit from the Object Repository.

InTransaction

InTransaction: Boolean; read only

Derived from TDatabase InTransaction indicates whether a database transaction is in progress or not. Examine InTransaction at run-time to determine if a database transaction is currently in progress. InTransaction is true if a transaction is in progress, false otherwise.

The value of InTransaction cannot be changed directly. Calling StartTransaction sets InTransaction to true. Calling Commit or Rollback sets InTransaction to false.

KeepConnection

KeepConnection: Boolean; read or write

Derived from TDatabase KeepConnection specifies whether an application remains connected to a database even if no datasets are open. Use KeepConnection to specify whether an application remains connected to a database even if no datasets are currently open. When KeepConnection is true (the default) the connection is maintained. For connections to remote database servers, or for applications that frequently open and close datasets, set KeepConnection to true to reduce network traffic, speed up applications, and avoid logging in to the server each time the connection is reestablished.

When KeepConnection is false a connection is dropped when there are no open datasets. Dropping a connection releases system resources allocated to the connection, but if a dataset is later opened that uses the database, the connection must be reestablished and initialized.

Note: The KeepConnection setting for temporary database components created automatically as needed is determined by the KeepConnections property of TSession.

ProcessingHidden

ProcessingHidden: Boolean; read or write

Some modifications on large databases may take quite a long time (up to several minutes !). If ProcessingHidden is set to true a processingdialog with a progressbar is shown to inform the user about the processing state of the operations.

ReadOnly

ReadOnly: Boolean; read or write

Derived from TDatabase use ReadOnly to specify whether the database connection should allow the application to update the tables and other metadata in the database. Set ReadOnly before opening the database.

When ReadOnly is false (the default), the application can modify tables and database metadata (like indeces). When ReadOnly is true, applications can browse tables but cannot update them. The application is also prevented from creating or deleting metadata objects like tables and indeces.

Note: A ReadOnly property value of true should be used when accessing a database on a read-only storage medium, such as a CD.

Session

Session: TOKSDBESession; read only

Derived from TDatabase use Session to determine the session component that controls the database component. By default, a database component is associated with the default session, Session, that is automatically created for all database applications. To assign a database component to a different session in a multi-threaded application, specify the name of a different session component in the SessionName property.

SessionName

SessionName: String; read or write

Derived from TDatabase SessionName identifies the name of the session used by this database component. Use SessionName to specify the session with which a database component is associated. If SessionName is blank, a database component is automatically associated with the default session, Session.

To associate a database component with a different session in a database application SessionName must matches the SessionName property of an existing session component.

Methods

ChangeAdminPass

ChangeAdminPass(NewAdminPass: String; Newcry_agm: TOKSDBEcry_agm)

Use ChangeAdminPass to change the admin's password of the database. The Newcry_agm is the parameter for the encryption algorithm and must always be twofish_128. This is parameter is resered for further versions when more algorithms will be implemented.

Close

Close;

Derived from TCustomConnection call Close to close the database and it's connection to the databasefile on harddisk. Call Close to disconnect from the remote source of database information. Before the connection component is deactivated, all associated datasets are closed. Calling Close is the same as setting the ConnectedToDatabase property to false.

In most cases, closing a connection frees system resources allocated to the connection.

CloseDataSets

CloseDatasets;

Derived from TDatabase CloseDataSets closes all datasets associated with the database component without disconnecting from the database server. Call CloseDataSets to close all active datasets without disconnecting from the database server. Ordinarily, when an application calls Close, all datasets are closed, and the connection to the database server is dropped. Calling CloseDataSets instead of Close ensures that an application can close all active datasets without having to reconnect to the database server at a later time.

Commit

Commit(goFlushBuffers: Boolean);

Derived from TDatabse Commit permanently stores updates, insertions, and deletions of data associated with the current transaction, and ends the current transactions.

Set parameter goFlushBuffers to true posts all changes that have been written to the record buffer. goFlushBuffers is called to cause the dataset to post all pending changes to the database, including any cached updates.

Call Commit to permanently store to the database server all updates, insertions, and deletions of data associated with the current transaction and then end the transaction. The current transaction is the last transaction started by calling StartTransaction.

Note: Before calling Commit, an application may check the status of the InTransaction property. If an application calls Commit and there is no current transaction, an exception is raised.

Create

Create(AOwner: Tcomponent);

Derived from TDatabase Create creates an instance of a TOKSDBEDatabase component. Call Create to instantiate a database component at runtime. An application can create a database component in order to control the component’s existence and set its properties and events, or an application can let Delphi create temporary database components as needed at runtime.

AOwner is the component that is responsible for freeing the database instance. It becomes the value of the Owner property.

CreateDatabase

CreateDatabase;

Creates a new Database on Harddisk. Before calling CreateDatabase specify at designtime or runtime the Databasefilename property. The default filenameextension is „.odb“. Ensure at runtime that the desired databasefilename doesn't exist by using the property Exists. Otherwise the existing database will be overwritten without any prompt!

DeleteDatabase

DeleteDatabase;

Calling DeleteDatabase will remove the database specified it's DatabaseFileName property from harddisk without any prompt. Ensure that the user will get a prompt by a warningdialog. Before calling DeleteDatabase the database must be closed by calling the close-method.

Destroy

Destroy;

Derived from TDatabase Destroy destroys the instance of a database component. Do not call Destroy directly in an application. Instead, call Free, which verifies that the database reference is not nil before calling Destroy. At runtime ensure that all modifications on the database and it's contents will be written to database if necessary.

FlushBuffers

```
FlushBuffers;
```

Derived from TBDEDataSet FlushBuffers posts all changes that have been written to the record buffer. Call FlushBuffers to cause the dataset to post all pending changes to the database, including any cached updates. Use FlushBuffers instead of CheckBrowseMode if it is important that cached record buffers are posted.

GetDBFileConnCount

```
GetDBFileConnCount: Integer;
```

Call GetDBFileConnCount to retrieve the amount of files connected to the database.

GetDBTables

```
GetDBTables(AllSessionsList: TStrings);
```

In the style of TDatabase call GetDBTables to retrieve a stringlist of all tables in all database sessions. Call GetDBTables to retrieve a list of tables in the associated database.

AllSessionsList is a TStrings descendant that receives the table names. Any existing strings are deleted from the list before GetDBTables adds the names of all tables in the database.

Open

```
Open;
```

Call Open method at runtime to open the database specified by it's databasefilename and databasename. Open sets the ConnectedToDatabase property to true.

RenameDatabase

```
RenameDatabase(NewDatabaseFileName: String);
```

Call RenameDatabase method to set the databasefilename to a new filename. Ensure that the new databasefilename does not exist on harddisk by using the property Exists. RenameDatabase will overwrite any existing database on harddisk without any propmt !

Rollback

```
Rollback;
```

Derived from TDatabase Rollback cancels all updates, insertions, and deletions for the current transaction and ends the transaction. Call Rollback to cancel all updates, insertions, and deletions for the current transaction and to end the transaction. The current transaction is the last transaction started by calling Start-Transaction.

Note: Before calling Rollback, an application may check the status of the InTransaction property. If an application calls Rollback and there is no current transaction, an exception is raised.

StartTransaction

StartTransaction;

Derived from TDatabase StartTransaction begins a new transaction against the database server. Before calling StartTransaction, an application should check the status of the InTransaction property. If InTransaction is true, indicating that a transaction is already in progress, a subsequent call to StartTransaction without first calling Commit or Rollback to end the current transaction raises an exception.

Updates, insertions, and deletions that take place after a call to StartTransaction are held by the server until an application calls Commit to save the changes or Rollback is to cancel them.

TableIsExisting

TableIsExisting(TableName: String) : Boolean;

Call TableIsExisting at runtime to ensure that a tablename in database does not exist before creating a new table otherwise the existing table with all it's contents will be overwritten without any prompt !

TruncateDatabase

TruncateDatabase;

In some cases modifications on databases will cause empty space not used any more in databases or it's datasets. Call TruncateDatabase at runtime to truncate this unused space.

Events

AfterChangeAdminPass

AfterChangeAdminPass: TNotifyEvent

Write an AfterChangeAdminPass event handler to take specific action immediately after an application changes the admin's password. AfterChangeAdminPass is called after the admin's password is changed by calling ChangeAdminPass.

BeforeChangeAdminPass

BeforeChangeAdminPass: TnotifyEvent

Write an AfterChangeAdminPass event handler to take specific action immediately before an application changes the admin's password. BeforeChangeAdminPass is called before the admin's password is changed by calling ChangeAdminPass.

OnAdminPass

OnAdminPass: TOKSDBEAdminPassEvent

Write an OnAdminPass event handler to take specific action as an application changes the admin's password. OnAdminPass is part of the ChangeAdminPass process.

OnChangeAdminPassProcessing

OnChangeAdminPassProcessing: TOKSDBEDatasetProcessingEvent

Write an OnChangeAdminPassProcessing event handler to take specific action while an application changes the admin's password. On large databases changing the admin's password may take quite several minutes. The OnChangeAdminPassProcessing events is used at runtime to give the user the possibilty to cancel the progress or showing some specific progressdialogs.

TOKSDBETable

Properties

Active

Active: Boolean; read or write

Derived from TDataset use Active to determine or set whether a dataset (TOKSDBETable or TOKSDBEQuery) is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true

- Generates a BeforeOpen event.
- Sets the dataset state to dsBrowse.
- Establishes a way to fetch data (typically by opening a cursor).
- Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1) Triggers a BeforeClose event.
2) Sets the State property to dsInactive.
3) Closes the cursor.
4) Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

AutoCalcFields

AutoCalcFields: Boolean; read or write

Derived from TDataset AutoCalcFields determines when the OnCalcFields event is triggered and when lookup field values are calculated. Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

- The dataset is opened.
- The dataset is put into dsEdit state.
- Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

- The dataset is opened.
- The dataset is put into dsEdit state.
- A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

CountDBFieldKeys

CountDBFieldKeys: Integer; read or write

In the style of TTable CountDBFieldKeys specifies the number of fields to use when conducting a partial key search on a multi-field key. Use CountDBFieldKeys to limit a search based on a multi-field key to a consecutive subset of those fields. For example, if the primary key for a dataset consists of three fields, a partial-key search can be conducted using only the first field in the key by setting CountDBFieldKeys to 1. If CountDBFieldKeys is 0, the dataset searches on all fields in the key.

Note: Searches are only conducted based on consecutive key fields beginning with the first field in the key. For example if a key consists of three fields, an application can set CountDBFieldKeys to 1 to search on the first field, 2 to search on the first and second fields, or 3 to search on all fields. By default CountDB-FieldKeys is initially set to include all fields in a search.

DatabaseName

DatabaseName: String; read or write

Derived from TDatabase DatabaseName specifies the name of the database to associate with the database component. Use DatabaseName to specify the name of the database to use with a database component. If DatabaseName is the same as an existing.

Note: Attempting to set DatabaseName when the Connected property is true raises an exception.

Exclusive

Exclusive: Boolean; read or write

Derived from TTable Exclusive enables an application to gain sole access to a TOKSDBETable. Use Exclusive to prevent other applications from accessing a TOKSD-BETable while it is open in this application. Before opening the table, set Exclusive to true. A table must be closed before changing the Exclusive property.

When Exclusive is true, then when the application successfully opens the table, no other application can access it. If the table for which the application has requested exclusive access is already in use by another application, an exception is raised. To handle such exceptions, wrap the code that opens the table in a try..catchexcept block.

Do not set Exclusive to true at design time if you also set the Active property to true at design time. In this case an exception is raised because the table is already in use by the IDE.

FieldDefs

FieldDefs: TOKSDBEFieldDefs; read or write

Derived from TDataset FieldDefs points to the list of field definitions for the dataset. FieldDefs lists the field definitions for a dataset. While an application can examine FieldDefs to explore the field definitions for a dataset, it should not change these definitions unless creating a new table.

To access fields and field values in a dataset, use the Fields, AggFields, and FieldValues properties, and the FieldsByName method.

Filter

Filter: String; read or write

Derived from TBDEDataSet Filter specifies the text of the current filter for a dataset. Use Filter to specify a dataset filter. When filtering is applied to a dataset, only those records that meet a filter's conditions are available to an application. Filter contains a string describing the filter condition. For example, the following filter condition displays only those records where the State field is 'CA' or 'MA':   State = 'CA' or State = 'MA'

When a filter is set, Blank records do not appear unless explicitly included in the filter. For example: State <> 'CA' or State is NULL

Filter expressions on remote SQL tables and on client datasets support field comparisons. For example: Field1 > Field2

Field comparisons are not supported.

The FilterOptions property controls case sensitivity and filtering on partial comparisons.

Tip: Applications can set Filter at runtime to change the filtering condition for a dataset (for example, in response to user input).

Filtered

Filtered: Boolean; read or write

Derived from TBDEDataSet Filtered specifies whether filtering is active for a dataset. Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is True, then filtering is active. Otherwise Filtered is False. To apply filter conditions specified in the Filter property or the OnFilterRecord event handler, set Filtered to True.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time the record is retrieved from the dataset while the filter is in effect, the record may seem to disappear. If that happens, the next record that passes the filter condition becomes the current record.

FilterOptions

FilterOptions: TfilterOptions;

Derived from TBDEDataSet FilterOptions specifies whether filtering is case insensitive, and whether partial comparisons are permitted when filtering records. Set FilterOptions to specify whether filtering is case insensitive when filtering on string or character fields, and whether partial comparisons for matching filter conditions is allowed.

By default, FilterOptions is set to an empty set. For filters based on string fields, include foCaseInsensitive in FilterOptions to catch all variations on a string regardless of capitalization.

To prevent partial string comparisons, include foNoPartialCompare in Filter-Options.

Note: To filter strings bases on partial comparisons, exclude foNoPartialCompare from FilterOptions and use an asterisk as a wildcard. For example: State = 'M*'

IndexDBFieldCount

IndexDBFieldCount: Integer; read only

In the sytle of TCustomClientDataSet IndexDBFieldCount indicates the number of fields that make up the current index. Check IndexFieldCount to determine how many fields are used by the current index for the dataset.

IndexDBFieldNames

IndexDBFieldNames: String; read or write

In the sytle of TCustomClientDataSet IndexDBFieldNames lists the fields to use as an index. Use IndexFieldNames as an alternative method of specifying the index to use for a client dataset. Specify the name of each field on which to index the da-taset, separating names with semicolons. Ordering of field names is significant.

Indeces added using IndexDBFieldNames do not support grouping or maintained aggre-gates.

Tip: Use IndexDBFieldNames to create sort orders on the fly at runtime.

Note: The IndexDBFieldNames and IndexName properties are mutually exclusive. Setting one clears the other.

IndexDBFields

IndexDBFields[Index: Integer]: TField; read or write

In the sytle of TCustomClientDataSet IndexDBFields specifies the fields associated with the current index. IndexDBFields is a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexDBFields[0], the second is IndexDBFields[1], and so on.

Note: Do not set IndexDBFields directly. Instead use the IndexDBFieldNames proper-ty to order datasets on the fly at runtime.

IndexDefs

IndexDefs: TIndexDefs;

Derived from TCustomClientDataSet IndexDefs contains information about the indexes for a client dataset. Examine IndexDefs for index information. IndexDefs maintains an array of TIndexDef items, each of which describes an available index for the dataset.

IndexName

IndexName: String; read or write

Derived from TCustomClientDataSet IndexName identifies an index for the client dataset. Use IndexName to specify an alternative index for a client dataset. If IndexName is empty, the dataset's sort order is based on the IndexDBFieldNames property or on its default ordering in the data packet. Default ordering is determined by the predefined index, DEFAULT_ORDER.

If IndexName contains a valid index name, then that index is used to determine sort order of records.

Note: IndexDBFieldNames and IndexName are mutually exclusive. Setting one clears the other.

KeyExclusive

KeyExclusive: Boolean; read or write

Derived from TCustomClientDataSet KeyExclusive specifies the upper and lower boundaries for a range. Use KeyExclusive to specify whether a range includes or excludes the records that match its specified starting and ending values. By default, KeyExclusive is false, meaning that matching values are included.

To restrict a range to those records that are greater than the specified starting value and less than the specified ending value, set KeyExclusive to true.

MasterDatasetFields

MasterDatasetFields: String; read or write

In the style of TCustomClientDataSet MasterDatasetFields names one or more fields in a master table to link with corresponding fields in this dataset in order to establish a master-detail relationship. Use MasterFields after setting the MasterSource property to specify the names of one or more fields to use in establishing a detail-master relationship between this dataset and the one specified in MasterSource.

MasterDatasetFields is a string containing one or more field names in the master table. Separate field names with semicolons. Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this dataset for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

MasterSource

MasterSource: TDataSource; read or write

Derived from TCustomClientDataSet MasterSource specifies a data source component for the master dataset when establishing a detail-master relationship between this dataset and another one. Use MasterSource to specify the name of the data source component whose DataSet property identifies a dataset to use as a master table in establishing a detail-master relationship between this dataset and another one.

At design time choose an available data source from the MasterSource property’s drop-down list in the Object Inspector. After setting the MasterSource property, specify which fields to use in the master table by setting the MasterDatasetFields property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this data-set for display.

Note: At design time, use the Field Link designer to establish the master-detail relationship between two datasets.

Tip: MasterSource establishes a master/detail relationship using the traditional linked cursor approach. Alternately, client datasets can participate in master/de-tail relationships using nested datasets, where the detail dataset is stored with the master table’s data. To assign a client dataset’s data as the value of a nes-ted detail, use the DataSetField property.

ReadOnly

ReadOnly: Boolean; read or write

Derived from TTable ReadOnly specifies whether a table is read-only for this app-lication. Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the table. By default, ReadOnly is false, meaning users can po-tentially alter a table’s data.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a table. Other factors, such as insufficient SQL privileges for the application or its current user may prevent successful alterations.

To guarantee that users cannot modify or add data to a table,

1) Set the Active property to false.
2) Set ReadOnly to true.

When ReadOnly is true, the table’s CanModify property is false.

SessionName

SessionName: String; read or write

Derived from TDBDataSet SessionName identifies the name of the session with which this dataset is associated. Use SessionName to determine the session with which a dataset component is associated. SessionName is automatically set to the name of the SessionName property of the database component with which a dataset component is associated. If SessionName is blank, a dataset component is automatically asso-ciated with the default session, Session.

To associate a dataset component with a different session, SessionName must match the SessionName property of an existing session component that is also used by the database component with which this dataset is associated.

StoreDefs

StoreDefs: Boolean; read or write

Derived from TDBDataSet StoreDefs indicates whether field and index definitions are saved with the client dataset. If StoreDefs is true, the client dataset’s index and field definitions are stored with the client dataset when the form or data module that contains it is saved. Setting StoreDefs to true makes the CreateDataSet method into a one-step procedure that creates fields and indexes at runtime.

TableName

TableName: String; read or write

TableName indicates the name of the database table that this component encapsulates. Use TableName to specify the name of the database table this component encapsulates. At design time select a valid table name from the TableName drop-down list in the Object Inspector.

Note: To set TableName, the Active property must be false.

TbAlterFieldDefs

TbAlterFieldDefs: TOKSDBEAdvFieldDefs; read or write

TbAlterFieldDefs containes advanced field definitions. Advanced field definitions are such as „RTFMemo“. TbAlterFieldDefs represents the structure for creating a new table or restructuring an existing table structure and must be set before restructuring the table.

TbAlterIndexDefs

TbAlterIndexDefs: TOKSDBEAdvIndexDefs; read or write

TbAlterFieldDefs containes the new indeces for the new table structure. TbAlterIndexDefs represents all indeces for creating a new table or restructuring an existing table structure and must be set before restructuring the table.

UseTempMemoryProcessing

UseTempMemoryProcessing: Boolean; read or write

Setting UseTempMemoryProcessing to true will affect all operations done in a database to a temporary table. This table is not stored in the database and will be lost at closing the database. It's recommended to use only physical tables for datastorage. Otherwise you have to ensure that temporary tables contents are stored to a physical table by creating a new table or modifying an existing table and copy data to them from temporary table.

Methods

AddDatasetIndex

```
AddDatasetIndex(const Name, Fields: String; Options: TIndexOptions;
                const DescFields: String = ''; const CaseInsFields: String = '');
```

Derived from TTable AddDatasetIndex creates a new index for the table. Call Add-Index to create a new index for the already-existing table associated with a data-set component. The index created with this procedure is added to the database ta-ble underlying the dataset component.

Name is the name of the new index.

Fields contains the names of the fields on which the new index will be based. If more than one field is used, separate the field names in the list with semicolons.

Options is a set of attributes for the index. The Options parameter may contain any one, multiple, or none of the TIndexOptions constants: ixPrimary, ixDescen-ding and ixCaseInsensitive.

DescFields is a string containing a list of field names, separated by semicolons. The fields specified in DescFields are the fields in the new index for which the ordering will be descending. Fields in the index definition but not in the Desc-Fields list use the default ascending ordering. It is possible that a single index can have fields using both ascending and descending ordering.

CaseInsFields is a string containing a list of field names, separated by semico-lons. The fields specified in CaseInsFields are the fields in the new index for which the ordering will be case insensitive. Fields in the index definition but not in the CaseInsFields list use the default case sensitivity ordering. It is possible that a single index can have fields using both case sensitiv1ite and case insensitivite ordering.

BatchMove

```
BatchMove(SrcDataset: TOKSDBEDataSet; TypeOfBatchMove: TOKSDBEBatchMoveType);
```

Derived from TTable BatchMove moves records from a dataset into this table. Call BatchMove to

- Copy records from another table into this table.
- Update records in this table that occur in another table.
- Append records from another table to the end of this table.
- Delete records in this table that occur in another table.

SrcDataset is a dataset component containing the records to import or (if dele-ting) match. The TypeOfBatchMove parameter indicates what operation to perform (copy, replace, append, or delete). This table is the destination of the batch operation.

CopyTable

```
CopyTable(NewTableName: string; DestDatabaseFileName: string='';
          DestDatabaseAdminPass: string ='');
```

CopyTable copies a table from one database to another.

NewTableName is the name of the destination table in destination database. DestDatabaseFileName is the name of the destination database. DestDatabaseAdminPass is the admins's password of the destination database.

CreateTable

```
CreateTable;
```

Derived from TTable CreateTable builds a new table using new structure information. Call CreateTable at runtime to create a table using this dataset’s current definitions. If the table already exists, CreateTable overwrites the table’s structure and data. To avoid overwriting an existing table, check the Exists property before calling CreateTable.

If the FieldDefs property contains values, these values are used to create field definitions. Otherwise the Fields property is used. One or both of these properties must contain values in order to create a database table.

If the IndexDefs property contain values, these values are used to create indexes on the table.

DeleteAllDatasetIndeces

DeleteAllDatasetIndeces removes all Indeces from a Dataset such as TOKSDBETable or TOKSDBEQuery.

DeleteIndex

```
DeleteIndex(const Name: string);
```

Call DeleteIndex to remove an index for a table. Name is the name of the index to delete.

Note: To delete an index, an application must first open the table for exclusive access.

DeleteTable

```
DeleteTable;
```

Derived from TTable call DeleteTable to delete an existing database table associated with the table component through its DatabaseName and TableName properties. A table must be closed before it can be deleted.

Warning: Deleting a table erases any data the table contains and destroys the table’s structure information.

EmptyTable

```
EmptyTable;
```

Derived from TTable EmptyTable deletes all records from the table. The EmptyTable method deletes all records from the database table specified by the DatabaseName and TableName properties. If fieldtype Autoinc is applied, the counter will be reset to default value set by Default.

ExportTable

ExportTable(DestinationTable: TDataset; CreateTablePointer: TProcedure): Boolean;

ExportTable copies the data (only) of an table to another table component. CreateTablePointer points to the adress of the procedure for creating the new table. TProcedure is used as the type for generic procedures that do not have arguments.

FindKey

FindKey(const KeyValues: array of const): Boolean;

Derived from TTable FindKey searches for a record containing specified field values. Call FindKey to search for a specific record in a dataset. KeyValues contains a comma-delimited array of field values, called a key. Each value in the key can be a literal, a variable or nil. If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be null. KeyValues_Size specifies the index of the last value in KeyValues (one less than the total number of values supplied). The key must always be an index, which can be specified in the IndexName property. If IndexName is empty oksDBEngine raises an exception. If the search is successful, FindKey positions the cursor on the matching record and returns true. Otherwise the cursor is not moved, and FindKey returns false.

FindKeyNearest

FindKeyNearest(const KeyValues: array of const);

In the style of TTable FindKeyNearest moves the cursor to the record that most closely matches a specified set of key values. Call FindNearest to move the cursor to a specific record in a dataset or to the first record in the dataset that is greater than the values specified in the KeyValues parameter. KeyValues contains a comma-delimited array of field values, called a key. Each value in the key can be a literal, a variable, or nil. If the number of values passed in KeyValues is less than the number of columns in the index used for the search, the missing values are assumed to be null. KeyValues_Size specifies the index of the last value in KeyValues (one less than the total number of values supplied). The key must always be an index, which can be specified in the IndexName property. If IndexName is empty oksDBEngine raises an exception.

FindNearest positions the cursor either on a record that exactly matches the search criteria, or on the first record whose values are greater than those specified in the search criteria. If there are no records that match or exceed the specified criteria, FindNearest positions the cursor on the first record in the table. KeyExclusive affects the boundary conditions of ranges and will affect the record selected by FindNearest.

GotoKey

GotoKey: Boolean;

Derived from TTable GotoKey moves the cursor to a record specified by the current key. Use GotoKey to move to a record specified by key values assigned with previous calls to SetKey or EditKey and actual search values indicated in the Fields property.

If GotoKey finds a matching record, it positions the cursor on the record and returns true. Otherwise the current cursor position remains unchanged, and GotoKey returns false. The key must always be an index, which can be specified in the IndexName property. If IndexName is empty oksDBEngine raises an exception.

GotoKeyNearest

GotoKeyNearest;

In the style of TTable GotoKeyNearest moves the cursor to the record that most closely matches the current key. Call GotoKeyNearest to position the cursor on the record that is either the exact record specified by the current key values in the key buffer, or on the first record whose values exceed those specified. If there are no records that match or exceed the specified criteria, GotoNearest positions the cursor on the first record in the table.

Before calling GotoKeyNearest, an application must specify key values by calling SetKey or EditKey to put the dataset is dsSetKey state, and then use FieldByName to populate the key buffer property with search values.

The key must always be an index, which can be specified in the IndexName property. If IndexName is empty, oksDBEngine raises an exception.

Note: KeyExclusive determines which records are considered part of a search range.

ImportTable

ImportTable(SourceTable: TDataset): Boolean;

ImportTable copies the data (only) of an another table component into this table. SourceTable is is the dataset to copy the contents from.

Post

Post;

Derived from TBDEDataSet Post writes a modified record to the database. Call Post to write a modified record to the database. Dataset methods that change the dataset state, such as Edit, Insert, or Append, or that move from one record to another, such as First, Last, Next, and Prior automatically call Post.

ProduceSQL

ProduceSQL(FileName: String);

Call ProduceSQL to generate a file on harddisk specified by FileName containing the whole table's contents and data as a SQL-Statement. This feature is well known from other SQL-Databases like mySQL.

RangeCancel

```
RangeCancel;
```

In the style of TTable RangeCancel removes any ranges currently in effect for the table. Call RangeCancel to remove a range currently applied to a table. Canceling a range reenables access to all records in the dataset.

RangeEndEdit

```
RangeEndEdit;
```

In the style of TTable RangeEndEdit enables changing the ending value for an existing range. Call RangeEndEdit to change the ending value for an existing range. To specify an end range value, call FieldByName after calling EditRangeEnd. After assigning a new ending value, call ApplyRange to activate the modified range. RangeEndEdit works only on indexed fields.

RangeStartEdit

```
RangeStartEdit;
```

In the style of TTable RangeStartEdit enables changing the starting value for an existing range. Call RangeStartEdit to change the starting value for an existing range. To specify a start range value, call FieldByName after calling EditRangeSt-art. After assigning a new ending value, call ApplyRange to activate the modified range. RangeStartEdit works only on indexed fields.

RenameField

```
RenameField(FieldName, NewFieldName: String);
```

Call RenameField to rename the field specified by FieldName to NewFieldName.

RenameTable

```
RenameTable(NewTableName: String);
```

Derived from TTable RenameTable renames the table associated with this table component. Call RenameTable to give a new name to the table underlying this table component.

RetrieveLastAutoInc

```
RetrieveLastAutoInc(FieldName: String): Int64;
```

Call RetrieveLastAutoInc to retrieve the last Autoinc value generated for the specified FieldName.

SetKey

```
SetKey;
```

Derived from TTable SetKey enables setting of keys and ranges for a dataset prior to a search. Call SetKey to put the dataset into dsSetKey state and clear the current contents of the key buffer. The FieldByName method can then be used to supply a new set of search values prior to conducting a search.

Note: To modify an existing key or range, call EditKey.

SetRange

SetRange(const StartValues, EndValues: array of const);

Derived from TTable SetRange sets the starting and ending values of a range, and applies it. Call SetRange to specify a range and apply it to the dataset. The new range replaces the currently applied range, if any.

StartValues indicates the field values that designate the first record in the range. StartValues_Size specifies the index of the last value in StartValues (one less than the total number of values).

EndValues indicates the field values that designate the last record in the range. EndValues_Size specifies the index of the last value in EndValues (one less than the total number of values).

SetRange combines the functionality of SetRangeStart, SetRangeEnd, and ApplyRange in a single procedure call. SetRange performs the following functions:

1) Puts the dataset into dsSetKey state.
2) Erases any previously specified starting range values and ending range values.
3) Sets the start and end range values.
4) Applies the range to the dataset.

If either StartValues or EndValues has fewer elements than the number of fields in the current index, then the remaining entries are set to NULL.

After a call to SetRange, the cursor is left on the first record in the range.

SetRange works only on indexed fields.

TbAlterTable

TbAlterTable: Boolean;

Set TbAlterTable to true to apply the table's restructuring process. Before setting TbAlterTable to true TbAlterFieldDefs and TbAlterIndexDefs must be set. It's recommended to write event handlers for user information, because altering table's structure may take several minutes to perform on large databases.

Events

AfterBatchMove

AfterBatchMove: TOKSDBEDatasetNotifyEvent;

Write an AfterBatchMove event handler to take specific action immediately after an application has performed the BatchMove method.

AfterCancel

AfterCancel: TDataSetNotifyEvent;

Occurs after an application completes a request to cancel modifications to the active record. Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

AfterClose

AfterClose: TDataSetNotifyEvent;

Occurs after an application closes a dataset. Write an AfterClose event handler to take specific action immediately after an application closes a dataset. AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

AfterDatasetCopy

AfterDatasetCopy: TOKSDBEDatasetNotifyEvent;

Occurs after an application has finished the CopyTable method. Write an AfterDatasetCopy event handler to take specific action immediately after an application has finished the CopyTable method.

AfterDelete

AfterDelete: TDataSetNotifyEvent;

Occurs after an application deletes a record. Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

AfterEdit

AfterEdit: TDataSetNotifyEvent;

Occurs after an application starts editing a record. Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

AfterExport

AfterExport: TOKSDBEDatasetNotifyEvent;

Occurs after an application has finished the ExportTable method. Write an AfterExport event handler to take specific action immediately after the ExortTable method has finished.

AfterImport

AfterImport: TOKSDBEDatasetNotifyEvent;

Occurs after an application has finished the ImportTable method. Write an AfterImport event handler to take specific action immediately after the ImportTable method has finished.

AfterInsert

AfterInsert: TDataSetNotifyEvent;

Occurs after an application inserts a new record. Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

AfterOpen

AfterOpen: TDataSetNotifyEvent;

Occurs after an application completes opening a dataset and before any data access occurs. Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

AfterPost

AfterPost: TDataSetNotifyEvent;

Occurs after an application writes the active record to the database or change log and returns to browse state. Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

AfterRefresh

AfterRefresh: TDataSetNotifyEvent;

Occurs after an application refreshes the data in the dataset. Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh method.

AfterScroll

AfterScroll: TDataSetNotifyEvent;

Occurs after an application scrolls from one record to another. Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

AfterTBAlter

Occurs after an application has finished the TbAlter method. Write an AfterScroll event handler to take specific action immediately after an application has finished altering a table's structure by the TbAlter method.

BeforeBatchMove

BeforeBatchMove: TOKSDBEDatasetNotifyEvent;

Occurs before an application executes a the BatchMove method. Write a BeforeBatchMove event to take specific action before an application starts the BatchMove method for moving datasets.

BeforeCancel

BeforeCancel: TDataSetNotifyEvent;

Occurs before an application executes a request to cancel changes to the active record. Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user’s changes in an undo buffer.

BeforeClose

BeforeClose: TDataSetNotifyEvent;

Occurs immediately before the dataset closes. Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

BeforeDatasetCopy

BeforeDatasetCopy: TOKSDBEDatasetNotifyEvent;

Occurs immediately before an application starts the CopyTable method. Write a BeforeDatasetCopy event to take specific action before an application starts the CopyTable method.

BeforeDelete

BeforeDelete: TDataSetNotifyEvent;

Occurs before an application attempts to delete the active record. Write a Before-Delete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

BeforeEdit

BeforeEdit: TDataSetNotifyEvent;

Occurs before an application enters edit mode for the active record. Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

BeforeExport

BeforeExport: TOKSDBEDatasetNotifyEvent;

Occurs before an application starts the ExportTable method. Write a BeforeExport event handler to take specific action before an application starts exporting table called by the ExportTable method.

BeforeImport

BeforeImport: TOKSDBEDatasetNotifyEvent;

Occurs before an application starts the ImportTable method. Write a BeforeImport event handler to take specific action before an application starts importing table called by the ImportTable method.

BeforeInsert

BeforeInsert: TDataSetNotifyEvent;

Occurs before an application enters insert mode. Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

BeforeOpen

BeforeOpen: TDataSetNotifyEvent;

Occurs before an application executes a request to open a dataset. Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

BeforePost

BeforePost: TDataSetNotifyEvent;

Occurs before an application posts changes for the active record to the database or change log. Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation.

BeforeRefresh

BeforeRefresh: TDataSetNotifyEvent;

Occurs immediately before an application refreshes the data in the dataset. Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

BeforeScroll

BeforeScroll: TDataSetNotifyEvent;

Occurs before an application scrolls from one record to another. Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

BeforeTBAlter

BeforeTbAlter: TOKSDBEDatasetNotifyEvent;

Occurs before an application starts altering table's structure called by the TbAlter method. Write a BeforeTbAlter event handler to take specific action immediately before an application starts altering table's structure called by the TbAlter method. For example a warningdialog might be shown that indicates the user altering table's structure might take up to several minutes to process on large databases.

OnBatchMoveProcessing

OnBatchMoveProcessing: TOKSDBEDatasetProcessingEvent;

Occurs when an application starts the BatchMove process. Write an OnBatchMoveProcessing event handler to take specific action when an application starts the BatchMove method.

OnCalcFields

OnCalcFields: TDataSetNotifyEvent;

Occurs when an application recalculates calculated fields. Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

- A dataset is opened.
- A dataset is put into dsEdit state.
- A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

- Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

OnCopyProcessing

OnCopyProcessing: TOKSDBEDatasetProcessingEvent;

Occurs when an application starts copying a table from one database to another called by the CopyTable method. Write an OnCopyProcessing event handler to take specific action when an application starts copying a table from one database to another called by the CopyTable method.

OnDeleteError

OnDeleteError: TDataSetErrorEvent;

Occurs when an application attempts to delete a record and an exception is raised. Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

OnEditError

```
OnEditError: TDataSetErrorEvent;
```

Occurs when an application attempts to modify or insert a record and an exception is raised. Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

OnExportProcessing

```
OnExportProcessing: TOKSDBEDatasetProcessingEvent;
```

Occurs when an application starts exporting data of an table to another table component. Write an OnExportProcessing event handler when an applications starts exporting data of an table to another table component.

OnFilterRecord

```
OnFilterRecord: TFilterRecordEvent;
```

Occurs each time a different record in the dataset becomes the active record and filtering is enabled. Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. oksDBEngine initializes Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can’t be implemented using the Filter property. For example, using the Filter property, field comparisons are not supported against local tables, but an OnFilterRecord event handler can implement any criterion at all.

Tip: Be sure that any interactions between the Filter property and the OnFilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

OnImportProcessing

```
OnImportProcessing: TOKSDBEDatasetProcessingEvent;
```

Occurs when an application starts importing data of another table component. Write an OnImportProcessing event handler when an applications starts importing data of another table component called by the ImportTable method.

OnNewRecord

```
OnNewRecord: TDataSetNotifyEvent;
```

Occurs when an application inserts or appends a new dataset record. Write an OnNewRecord event handler to take specific actions as an application inserts or appends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

OnPostError

OnPostError: TDataSetErrorEvent;

Occurs when an application attempts to modify or insert a record and an exception is raised. Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

OnTBAlterProcessing

OnTbAlterProcessing: TOKSDBEDatasetProcessingEvent;

Occurs when an application starts altering table's structure called by the TbAlter method. Write an OnTbAlterProcessing event handler when an applications starts altering table's structure called by the TbAlter method. For example a processing-dialog might be shown that indicates the user the progress of altering table's structure on large databases.

OnUpdateRec

OnUpdateRec: TUpdateRecordEvent;

type TUpdateRecordEvent = procedure(DataSet: TDataSet; UpdateKind: TUpdateKind; var UpdateAction: TUpdateAction) of object;

Occurs when cached updates are applied to a record. Write an OnUpdateRec event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This handler is also useful for applications that require additional control over parameter substitution in update components.

DataSet is the name of the dataset to which updates are applied.

UpdateKind indicates whether the current update is the insertion of a record, the deletion of a record, or the modification of a record.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting.

The code in an OnUpdateRecord handler must not call any methods that make a different record the current one.

Note: As an alternate approach, application can use a client dataset and a provider component to handle cached updates. This alternative provides more support and greater control.

TOKSDBEQuery

Properties

Active

Active: Boolean; read or write

Derived from TDataset use Active to determine or set whether a dataset (TOKSDBETable or TOKSDBEQuery) is populated with data. When Active is false, the dataset is closed; the dataset cannot read or write data and data-aware controls can not use it to fetch data or post edits. When Active is true, the dataset can be populated with data. It can read data from a database or other source (such as a provider). Depending on the CanModify property, active datasets can post changes.

Setting Active to true

- Generates a BeforeOpen event.
- Sets the dataset state to dsBrowse.
- Establishes a way to fetch data (typically by opening a cursor).
- Generates an AfterOpen event.

If an error occurs while opening the dataset, dataset state is set to dsInactive, and any cursor is closed.

Setting Active to false:

1) Triggers a BeforeClose event.
2) Sets the State property to dsInactive.
3) Closes the cursor.
4) Triggers an AfterClose event.

An application must set Active to false before changing other properties that affect the status of a database or the controls that display data in an application.

Note: Calling the Open method sets Active to true; calling the Close method sets Active to false.

AffectedRows

AffectedRows: Integer; read only

In the style of TQuery AffectedRows returns the number of rows operated upon by the latest query execution. Inspect AffectedRows to determine how many rows were updated or deleted by the last query operation. If no rows were updated or deleted, AffectedRows has a value of zero. AffectedRows will have a value of –1 if the execution of the SQL statement could not be executed due to an error condition. This latter situation would typically follow the raising of an exception.

AutoCalcFields

AutoCalcFields: Boolean; read or write

Derived from TDataset AutoCalcFields determines when the OnCalcFields event is triggered and when lookup field values are calculated. Set AutoCalcFields to control when the OnCalcFields event is triggered to update calculated fields and when lookup fields are calculated.

A calculated field is one that derives its value from the values of one or more fields in the active record, sometimes with additional processing. Lookup fields are fields whose values come from a secondary dataset or lookup cache.

Note: Unidirectional datasets support calculated fields but not unidirectional fields. For unidirectional datasets, OnCalcFields only controls when calculated fields are updated.

When AutoCalcFields is true (the default), Lookup fields are recalculated and OnCalcFields is triggered when:

- The dataset is opened.
- The dataset is put into dsEdit state.
- Focus moves from one visual control to another, or from one column to another in a data-aware grid and modifications have been made to the record.

When AutoCalcFields is false, Lookup fields are recalculated and the OnCalcFields event occurs only when

- The dataset is opened.
- The dataset is put into dsEdit state.
- A record is retrieved from a database.

If an application permits users to change data, OnCalcFields is frequently triggered. In these cases an application may set AutoCalcFields to false to reduce the frequency with which the OnCalcFields event occurs and with which lookup values are fetched.

DatabaseName

DatabaseName: String; read or write

Derived from TDatabase DatabaseName specifies the name of the database to associate with the database component. Use DatabaseName to specify the name of the database to use with a database component. If DatabaseName is the same as an existing.

Note: Attempting to set DatabaseName when the Connected property is true raises an exception.

DataSource

DataSource: TDataSource; read or write

Derived from TQuery DataSource specifies the data source component from which to extract current field values to use with same-name parameters in the query’s SQL statement. Set DataSource to automatically fill parameters in a query with fields values from another dataset. Parameters that have the same name as fields in the other dataset are filled with the field values. Parameters with names that are not the same as fields in the other dataset do not automatically get values, and must be programmatically set. For example, if the SQLStatement of the TOKSDBEQuery contains the SQL statement below and the dataset referenced through DataSource has a CustNo field, the value from the current record in that other dataset is used in the CustNo parameter.

```
SELECT *
FROM Orders O
WHERE (O.CustNo = :CustNo)
```

DataSource must point to a TDataSource component linked to another dataset component; it cannot point to this query's data source component.

The dataset specified in DataSource must be created, populated, and opened before attempting to bind parameters. Parameters are bound by calling the query's Prepare method prior to executing the query.

Tip: DataSource is especially of use when creating a master-detail relationship between tables using a linked query. It is also of use to guarantee binding for parameters that are not already set in the Params property or through a call to the ParamByName method.

If the SQL statement used by a query does not contain parameters, or all parameters are bound by the application using the Params property or the ParamByName method, DataSource need not be assigned. The example below shows setting the DataSource property of OKSDBEQuery2 to the data source for OKSDBEQuery1, preparing OKSDBEQuery2, and activating OKSDBEQuery2.

```
with OKSDBEQuery2 do begin
  DataSource := DataSource1;
  Prepare;
  Open;
end;
```

If the SQL statement in the TOKSDBEQuery is a SELECT query, the query is executed using the new field values each time the record pointer in the other dataset is changed. It is not necessary to call the Open method of the TOKSDBEQuery each time. This makes using the DataSource property to dynamically filter a query result set useful for establishing Master-Detail relationships. Set the DataSource property in the Detail query to the TDataSource component for the Master dataset.

If the SQL statement uses other than a SELECT query (such as INSERT or UPDATE), the parameters with the same name as fields in the other dataset still get values, but the query must be explicitly executed each time the other dataset's record pointer moves. For example, the SQL statement below uses the INSERT statement and has the parameters CustNo and CompanyName.

```
INSERT INTO Customer
(CustNo, Company)
VALUES (:CustNo, :CompanyName)
```

Another dataset, OKSDBEQuery1 and DataSource1, has a CustNo field but no CompanyName field. If this dataset is used through the DataSource property, the CompanyName parameter must be programmatically assigned a value. Because OKSDBEQuery1 has a CustNo field and Query1 is referenced through the DataSource property, the CustNo parameter automatically receives a value.

```
with OKSDBEQuery2 do begin
  DataSource := DataSource1;
  ParamByName('CompanyName').AsString := Edit1.Text;
  Prepare;
  ExecSQL;
end;
```

If the SQL statement contains parameters with the same name as fields in the other dataset, do not manually set values for these parameters. Any values programmatically set, such as by using the Params property or the ParamByName method, will be overridden with automatic values. Parameters of other names must be programmatically given values. These parameters are unaffected by setting DataSource.

DataSource can be set at runtime or at design-time using the Object Inspector. At design-time, select the desired TDataSource from the drop-down list or type in the name.

Filter

Filter: String; read or write

Derived from TBDEDataSet Filter specifies the text of the current filter for a dataset. Use Filter to specify a dataset filter. When filtering is applied to a dataset, only those records that meet a filter's conditions are available to an application. Filter contains a string describing the filter condition. For example, the following filter condition displays only those records where the State field is 'CA' or 'MA':   State = 'CA' or State = 'MA'

When a filter is set, Blank records do not appear unless explicitly included in the filter. For example: State <> 'CA' or State is NULL

Filter expressions on remote SQL tables and on client datasets support field comparisons. For example: Field1 > Field2

Field comparisons are not supported.

The FilterOptions property controls case sensitivity and filtering on partial comparisons.

Tip: Applications can set Filter at runtime to change the filtering condition for a dataset (for example, in response to user input).

Filtered

Filtered: Boolean; read or write

Derived from TBDEDataSet Filtered specifies whether filtering is active for a dataset. Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is True, then filtering is active. Otherwise Filtered is False. To apply filter conditions specified in the Filter property or the OnFilterRecord event handler, set Filtered to True.

Note: When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time the record is retrieved from the dataset while the filter is in effect, the record may seem to disappear. If that happens, the next record that passes the filter condition becomes the current record.

FilterOptions

FilterOptions: TfilterOptions;

Derived from TBDEDataSet FilterOptions specifies whether filtering is case insensitive, and whether partial comparisons are permitted when filtering records. Set FilterOptions to specify whether filtering is case insensitive when filtering on string or character fields, and whether partial comparisons for matching filter conditions is allowed.

By default, FilterOptions is set to an empty set. For filters based on string fields, include foCaseInsensitive in FilterOptions to catch all variations on a string regardless of capitalization.

To prevent partial string comparisons, include foNoPartialCompare in FilterOptions.

Note: To filter strings bases on partial comparisons, exclude foNoPartialCompare from FilterOptions and use an asterisk as a wildcard. For example: State = 'M*'

ParamCheck

ParamCheck: Boolean; read or write

Derived from TQuery ParamCheck specifies whether the parameter list for a query is regenerated if the SQLStatement changes at runtime. Set ParamCheck to specify whether or not the Params property is cleared and regenerated if an application modifies the query’s SQLStatement at runtime. By default ParamCheck is true, meaning that the Params property is automatically regenerated at runtime. When ParamCheck is true, the proper number of parameters is guaranteed to be generated for the current SQL statement.

This property is useful for data definition language (DDL) statements that contain parameters as part of the DDL statement and that are not parameters for the TQuery. For example, the DDL statement to create a stored procedure may contain parameter statements that are part of the stored procedure. Set ParamCheck to false to prevent these parameters from being mistaken for parameters of the TQuery executing the DDL statement.

An application that does not use parameterized queries may choose to set ParamCheck to false, but otherwise ParamCheck should be true.

Params

Params: TParams; read or write

Derived from TQuery Params contains the parameters for a query’s SQL statement. Access Params at runtime to view and set parameter names, values, and data types dynamically (at design time use the collection editor for the Params property to set parameter information). Params is a zero-based array of TParams parameter records. Index specifies the array element to access.

Note: An easier way to set and retrieve parameter values when the name of each parameter is known is to call ParamByName. ParamByName cannot, however, be used to change a parameter’s data type or name.

Parameters used in SELECT statements cannot be NULL, but they can be NULL for UP-

DATE and INSERT statements.

Prepared

Prepared: Boolean; read or write

Derived from TQuery Prepared determines whether or not a query is prepared for execution. Examine Prepared to determine if a query is already prepared for execution. If Prepared is true, the query is prepared, and if Prepared is false, the query is not prepared. While a query need not be prepared before execution, execution performance is enhanced if the query is prepared beforehand, particularly if it is a parameterized query that is executed more than once using the same parameter values.

Note: An application can change the current setting of Prepared to prepare or unprepare a query. If Prepared is true, setting it to false calls the UnPrepare method to unprepare the query. If Prepared is false, setting it to true calls the Prepare method to prepare the query. Generally, however, it is better programming practice to call Prepare and UnPrepare directly. These methods automatically update the Prepared property.

ReadOnly

ReadOnly: Boolean; read or write

Derive from TCustomClientDataset ReadOnly specifies whether the client dataset is read-only for this application. Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the dataset. By default, ReadOnly is false, meaning users can potentially alter the dataset’s data.

To guarantee that users cannot modify or add data to a dataset, set ReadOnly to true.

When ReadOnly is true, the dataset’s CanModify property is false.

Note: Even if ReadOnly is false, users may not be able to modify or add data to a client dataset if it gets its data from a provider. The provider’s Options property can limit whether data can be edited, and if so, what types of edits are allowed.

RequestLive

RequestLive: Boolean; read or write

Derived from TQuery RequestLive requests an updatable query result set from the database. RequestLive is a request that a SELECT query returns an updatable (or live) or read-only result set from the database back-end. A false value for RequestLive (the default) causes the result set to always be read-only. No request is made to the database back-end to return an updatable result set. A true value for RequestLive is a request to database back-end to return an updatable result set. An updatable result set can be made available to the application’s user for direct data editing through visual data controls.

A true value for RequestLive is a request for an updatable result set. It does not guarantee that the database back-end will return an updatable result set. See the documentation for the specific database system used for the criteria needed for a live query result set. If the database cannot return an updatable result set, a read-only result set is returned instead. This is done automatically and usually without error. Some database systems may raise an exception on requests for live result sets that cannot be fulfilled. Check the success of the request by inspecting the dataset component's CanModify property.

If the SQL statement used in a TOKSDBEQuery is a general, multi-row SELECT statement, RequestLive can be used with either a true or a false value. For all other SQL statements issued from the TQuery, RequestLive should only be set to false. These other statements include noncursor single-row SELECT statements (more commonly found in stored procedure programming), noncursor Data Manipulation Language (DML) statements like UPDATE or DELETE, and Data Definition Language (DDL) statements like CREATE TABLE and DROP INDEX.

Note: All multi-table queries return read-only result sets. Other conditions may cause a query to return a read-only result set.

Note: Some SQL database systems require strict case-sensitivity for names of metadata objects in SQL statements. These SQL databases typically have problems with the way metadata names are passed by the oksDBEngine in live queries and an exception is raised. A workaround that helps in most cases is to enclose the metadata object references (like table and column names) in quotation marks.

SessionName

SessionName: String; read or write

Derived from TDBDataSet SessionName Identifies the name of the session with which this dataset is associated. Use SessionName to determine the session with which a dataset component is associated. SessionName is automatically set to the name of the SessionName property of the database component with which a dataset component is associated. If SessionName is blank, a dataset component is automatically associated with the default session, Session.

To associate a dataset component with a different session, SessionName must match the SessionName property of an existing session component that is also used by the database component with which this dataset is associated.

SQLBinary

SQLBinary: PChar; read or write

Derived from TQuery SQLBinary Points to the binary data stream that represents an SQL query statement or result set. Do not access SQLBinary. It is an internal binary data stream used by the query component to communicate directly with the oksDBEngine. To access or set the SQL statement that this query component executes, use the SQLStatement. To access or set the parameters used in a parameterized SQL statement, use the Params property.

SQLStatement

SQLStatement: TStrings; read or write

In the style of TQuery SQLStatement contains the text of the SQL statement to execute for the query. Use SQLStatement to provide the SQL statement that a query component executes when its ExecSQL or Open method is called. At design time the SQLStatement property can be edited by invoking the String List editor in the

Object Inspector.

The SQLStatement property may contain only one complete SQL statement at a time. In general, multiple “batch” statements are not allowed unless a particular server supports them.

The SQLStatement property can be used to access tables, using local SQL. The allowable syntax is a subset of SQL-92. See extern SQL help files for information on supported syntax.

The SQL statement in the SQLStatement property may contain replaceable parameters, following standard SQL-92 syntax conventions. Parameters are created and stored in the Params property.

Text

Text: PChar; read only

Derived from TQuery Text points to the actual text of the SQL query. Text is a read-only property that can be examined to determine the actual contents of SQL statement. For parameterized queries, Text contains the SQL statement with parameters replaced by the parameter substitution symbol (?) in place of actual parameter values.

In general there should be no need to examine the Text property. To access or change the SQL statement for the query, use the SQL property. To examine or modify parameters, use the Params property.

UseTempMemoryProcessing

UseTempMemoryProcessing: Boolean; read or write

Setting UseTempMemoryProcessing to true will affect all operations done in a database to a temporary query. This query is not stored in the database and will be lost at closing the database. It's recommended to use only physical query for datastorage. Otherwise you have to ensure that temporary tables contents are stored to a physical query by creating a new query or modifying an existing query and copy data to them from temporary query.

Methods

ExecSQL

ExecSQL;

Derived from TQuery ExecSQL executes the SQL statement for the query. Call ExecSQL to execute the SQL statement currently assigned to the SQL property. Use ExecSQL to execute queries that do not return a cursor to data (such as INSERT, UPDATE, DELETE, and CREATE TABLE).

Note: For SELECT statements, call Open instead of ExecSQL. ExecSQL prepares the statement in SQL property for execution if it has not already been prepared. To speed performance, an application should ordinarily call Prepare before calling ExecSQL for the first time.

GetDetailLinkFields

```
GetDetailLinkFields(MasterDatasetFields, DetailFields: TList);
```

In the style of TQuery GetDetailLinkFields Fills lists with the master and detail fields of the link. GetDetailLinkFields creates two lists of TFields from the master-detail relationship between two tables; one containing the master fields in MasterDatasetFields, and the other containing the detail fields in DetailFields.

ParamByName

```
ParamByName(const Value: string): TParam;
```

Derived from TQuery ParamByName accesses parameter information based on a specified parameter name. Call ParamByName to set or use parameter information for a specific parameter based on its name.

Value is the name of the parameter for which to retrieve information.

ParamByName is primarily used to set an parameter’s value at runtime. For example, the following statement retrieves the current value of a parameter called “Contact” into an edit box:

```
Edit1.Text := Query1.ParamByName('Contact').AsString;
```

Parameters used in SELECT statements cannot be NULL, but they can be NULL for UPDATE and INSERT statements.

Prepare

```
Prepare;
```

Derived from TQuery Prepare sends a query to the oksDBEngine and the server for optimization prior to execution. Call Prepare to have oksDBEngine and a remote database server allocate resources for the query and to perform additional optimizations. Preparing a query consumes some database resources, so it is good practice to call UnPrepare once the query is no longer needed.

If the query will only be executed once, the application does not need to explicitly call Prepare or UnPrepare. Executing an unprepared query generates these calls automatically. However, if the same query is to be executed repeatedly, it is more efficient to prevent these automatic calls by calling Prepare and UnPrepare explicitly.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

UnPrepare

```
UnPrepare;
```

Derived from TQuery UnPrepare frees the resources allocated for a previously prepared query. Call UnPrepare to free the resources allocated for a previously prepared query on the server and client sides.

Preparing a query consumes some database resources, so it is good practice for an application to unprepare a query once it is done using it. The UnPrepare method unprepares a query.

Note: When you change the text of a query at runtime, the query is automatically closed and unprepared.

Events

AfterCancel

AfterCancel: TDataSetNotifyEvent;

Occurs after an application completes a request to cancel modifications to the active record. Write an AfterCancel event handler to take specific action after an application cancels changes to the active record. AfterCancel is called by the Cancel method after it updates the current position, releases the lock on the active record if necessary, and sets the dataset state to dsBrowse. If an application requires additional processing before returning control to a user after a Cancel event, code it in the AfterCancel event.

AfterClose

AfterClose: TDataSetNotifyEvent;

Occurs after an application closes a dataset. Write an AfterClose event handler to take specific action immediately after an application closes a dataset. AfterClose is called after a dataset is closed and the dataset state is set to dsInactive.

AfterDelete

AfterDelete: TDataSetNotifyEvent;

Occurs after an application deletes a record. Write an AfterDelete event handler to take specific action immediately after an application deletes the active record in a dataset. AfterDelete is called by Delete after it deletes the record, sets the dataset state to dsBrowse, and repositions the current record.

AfterEdit

AfterEdit: TDataSetNotifyEvent;

Occurs after an application starts editing a record. Write an AfterEdit event handler to take specific action immediately after dataset enters edit mode. AfterEdit is called by Edit after it enables editing of a record, recalculates calculated fields, and calls the data event handler to process a record change.

AfterInsert

AfterInsert: TDataSetNotifyEvent;

Occurs after an application inserts a new record. Write an AfterInsert event handler to take specific action immediately after an application inserts a record. The Insert and Append methods generate an AfterInsert event after inserting or appending a new record.

AfterOpen

AfterOpen: TDataSetNotifyEvent;

Occurs after an application completes opening a dataset and before any data access occurs. Write an AfterOpen event handler to take specific action immediately after an application opens the dataset. AfterOpen is called after the dataset establishes access to its data and the dataset is put into dsBrowse state. For example, an AfterOpen event handler might check an ini file to determine the last record touched in the dataset the previous time the application ran, and position the dataset at that record.

AfterPost

AfterPost: TDataSetNotifyEvent;

Occurs after an application writes the active record to the database or change log and returns to browse state. Write an AfterPost event handler to take specific action immediately after an application posts a change to the active record. AfterPost is called after a modification or insertion is made to a record.

AfterRefresh

AfterRefresh: TDataSetNotifyEvent;

Occurs after an application refreshes the data in the dataset. Write an AfterRefresh event handler to take specific action immediately after an application has updated the records in the dataset. AfterRefresh is generated by calls to the Refresh method.

AfterScroll

AfterScroll: TDataSetNotifyEvent;

Occurs after an application scrolls from one record to another. Write an AfterScroll event handler to take specific action immediately after an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. AfterScroll is called after all other events triggered by these methods and any other methods that switch from record to record in the dataset.

BeforeCancel

BeforeCancel: TDataSetNotifyEvent;

Occurs before an application executes a request to cancel changes to the active record. Write a BeforeCancel event to take specific action before an application carries out a request to cancel changes. BeforeCancel is called by the Cancel method before it cancels a dataset operation such as Edit, Insert, or Delete.

An application might use the BeforeCancel event to record a user's changes in an undo buffer.

BeforeClose

BeforeClose: TDataSetNotifyEvent;

Occurs immediately before the dataset closes. Write a BeforeClose event to take specific action before an application closes a dataset. Calling Close or setting the Active property to false results in a call to the BeforeClose event handler.

BeforeDelete

BeforeDelete: TDataSetNotifyEvent;

Occurs before an application attempts to delete the active record. Write a BeforeDelete event handler to take specific action before an application deletes the active record. BeforeDelete is called by Delete before it actually deletes a record.

In Delphi, making use of this event an application might, for example, display a dialog box asking for confirmation before deleting the record. On denial of confirmation, the application could abort the deletion by calling the Abort procedure.

BeforeEdit

BeforeEdit: TDataSetNotifyEvent;

Occurs before an application enters edit mode for the active record. Write a BeforeEdit event handler to take specific action before an application enables editing of the active record. For example, an application that keeps a log of database edits could use the BeforeEdit event to record the edit request, time, and user before entering edit state.

BeforeInsert

BeforeInsert: TDataSetNotifyEvent;

Occurs before an application enters insert mode. Write a BeforeInsert event handler to take specific action before an application inserts or appends a new record. The Insert or Append method generates a BeforeInsert method before it sets the dataset into dsInsert state.

BeforeOpen

BeforeOpen: TDataSetNotifyEvent;

Occurs before an application executes a request to open a dataset. Write a BeforeOpen event handler to take specific action before an application opens a dataset for viewing or editing. BeforeOpen is triggered when an application sets the Active property to true for a dataset or an application calls Open.

BeforePost

BeforePost: TDataSetNotifyEvent;

Occurs before an application posts changes for the active record to the database or change log.Write a BeforePost event handler to take specific action before an application posts dataset changes. BeforePost is triggered when an application calls the Post method. Post checks to make sure all required fields are present, then calls BeforePost before posting the record.

An application might use BeforePost to perform validity checks on data changes before committing them. If it encountered a validity problem, it could call Abort to cancel the Post operation.

BeforeRefresh

BeforeRefresh: TDataSetNotifyEvent;

Occurs immediately before an application refreshes the data in the dataset. Write a BeforeRefresh event handler to take specific action immediately before an application updates the records in the dataset. BeforeRefresh is generated by calls to the Refresh method.

BeforeScroll

BeforeScroll: TDataSetNotifyEvent;

Occurs before an application scrolls from one record to another. Write a BeforeScroll event handler to take specific action immediately before an application scrolls to another record as a result of a call to the First, Last, MoveBy, Next, Prior, FindKey, FindFirst, FindNext, FindLast, FindPrior, and Locate methods. BeforeScroll is called before all other events triggered by these methods and any other methods that switch from record to record in the dataset.

OnCalcFields

OnCalcFields: TDataSetNotifyEvent;

Occurs when an application recalculates calculated fields. Write an OnCalcFields event handler to take specific action when an application recalculates calculated fields. A calculated field is one that derives its value from the values in one or more fields in the dataset, sometimes with additional processing.

OnCalcFields is triggered when:

- A dataset is opened.
- A dataset is put into dsEdit state.
- A record is retrieved from a database.

When the AutoCalcFields property is true, OnCalcFields is also triggered when:

- Focus moves from one visual control to another, or from one column to another is a data-aware grid control and modifications were made to the record.

Note: When the AutoCalcFields property is true, an OnCalcFields event handler should not modify the dataset (or a linked dataset if it is part of a master-detail relationship), because such modifications retrigger the OnCalcField event, leading to infinite recursion.

If an application permits users to change data, OnCalcFields is frequently triggered. To reduce the frequency with which OnCalcFields occurs, set AutoCalcFields to false.

Warning: When the dataset is the master table of a master-detail relationship, OnCalcFields occurs before detail sets have been synchronized with the master table.

OnDeleteError

OnDeleteError: TDataSetErrorEvent;

Occurs when an application attempts to delete a record and an exception is raised.Write an OnDeleteError event handler to handle exceptions that occur when an attempt to delete a record fails.

OnEditError

OnEditError: TDataSetErrorEvent;

Occurs when an application attempts to modify or insert a record and an exception is raised. Write an OnEditError event handler to handle exceptions that occur when an attempt to edit a record fails.

OnFilterRecord

OnFilterRecord: TFilterRecordEvent;

Occurs each time a different record in the dataset becomes the active record and filtering is enabled. Write an OnFilterRecord event handler to specify for each record in a dataset whether it should be visible to the application. To indicate that a record passes the filter condition, the OnFilterRecord event handler must set the Accept parameter to true. To exclude a record, set the Accept parameter to false. TOKSDBEQuery initializes Accept to true before calling the OnFilterRecord event handler.

Warning: Do not supply an OnFilterRecord event handler to a unidirectional dataset. Unidirectional datasets do not support filters, and assigning an OnFilterRecord event handler causes the dataset to raise an exception.

Filtering is enabled if the Filtered property is true. When an application is processing a filter, the State property for the dataset is dsFilter.

Use an OnFilterRecord event handler to filter records using a criterion that can’t be implemented using the Filter property.

Tip: Be sure that any interactions between the Filter property and the On-FilterRecord event handler do not result in an empty filter set when they are used simultaneously in an application.

OnNewRecord

OnNewRecord: TDataSetNotifyEvent;

Occurs when an application inserts or appends a new dataset record. Write an OnNe-wRecord event handler to take specific actions as an application inserts or ap-pends a new record. OnNewRecord is called as part of the insert or append process. An application might use the OnNewRecord event to set initial values for a record or as a way of implementing cascading insertions in related datasets.

OnPostError

OnPostError: TDataSetErrorEvent;

Occurs when an application attempts to modify or insert a record and an exception is raised. Write an OnPostError event handler to handle exceptions that occur when an attempt to post a record fails.

OnUpdateRec

OnUpdateRec: TUpdateRecordEvent;

type TUpdateRecordEvent = procedure(DataSet: TDataSet; UpdateKind: TUpdateKind; var UpdateAction: TUpdateAction) of object;

Occurs when cached updates are applied to a record. Write an OnUpdateRec event handler to process updates that cannot be handled by a single update component, such as implementation of cascading updates, insertions, or deletions. This hand-ler is also useful for applications that require additional control over parameter substitution in update components.

DataSet is the name of the dataset to which updates are applied.

UpdateKind indicates whether the current update is the insertion of a record, the deletion of a record, or the modification of a record.

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpda-teRecord is successful, it should set UpdateAction to uaApplied before exiting.

The code in an OnUpdateRecord handler must not call any methods that make a diffe-rent record the current one.

Note: As an alternate approach, application can use a client dataset and a provi-der component to handle cached updates. This alternative provides more support and greater control.

Orderform (Bestellformular)

Please fill out und send this orderform to support@olkrosoft.de. You will recieve an email with our banking account. As soon as we recieved the money we will send the software to your email-adress.

Bitte füllen Sie das Bestellformular vollständig aus und senden Sie dieses an support@olkrosoft.de. Sie erhalten anschliessend eine eMail mit unserer Bankverbindung. Sobald der Betrag auf unserem Konto gutgeschrieben wurde versenden wir die Sofware noch am gleichen Tag an Ihre eMail-Adresse.

First Name, Name (Vorname, Name)	
Company (Firma)	
Adress (Strasse)	
ZIP (Plz,Ort)	
Email	

Please not, that source isn't included in any license.
Bitte beachten Sie, dass der Quellcode nicht Bestandteil der Bestellung ist.

Please select (Bitte ankreuzen)	**Item (Artikel)** **oksDBEngine Prof. Edition 1.2**	**Unit Price (EP)**	**TotalP Price (GP)**
☐	Single developper license (Einzelplatzlizenz)	€ 89,00 GBP 60,00 USD 118,00	
☐	5 developper license (5 Entwicklerplätzer-Lizenz)	€ 299,00 GBP 200,00 USD 395,00	
☐	10 developper license (10 Entwicklerplätzer-Lizenz)	€ 449,00 GBP 300,00 USD 594,00	
☐	Site license (Unbeschränkte Lizenz)	€ 599,00 GBP 400,00 USD 792,00	
	Delivery via email (Versand per eMail)	€ 0,00 GBP 0,00 USD 0,00	0,00
	Total Amount **(Gesamtsumme)**	please enter currency (bitte Währung angeben)	

For other license options please contact us via email.
Für andere Lizenzoptionen kontaktieren Sie uns bitte per eMail.

Date (Datum)

Signature (Unterschrift)

Stichwortverzeichnis