Installing KADao

See **Readme.txt** for latest informaton about how to install KADao.

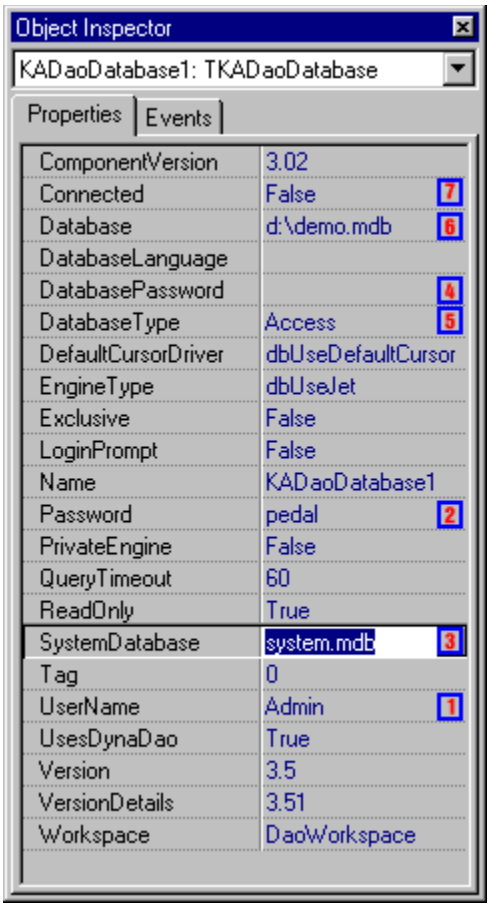
How to make an simple application using KADao as a Table component?

1. Place a KADaoDatabase component on the form
2. Set UserName Property
3. Set Password Property
4. If you use different system database then default set SystemDatabase property
5. If your entire database is password protected set DatabasePassword property
6. If your Database is not an MS Access database select apropriate database type from DatabaseType property
7. Choose Database filename or DSN using the Database property.
8. If you want to use an Access XX-97 database set **Version** property to **3.5**
9. If you want to use an Access 2000-2002 database set **Version** property to **3.6**
10. Set Connected property to True. **NOW YOU ARE CONNECTED TO THE DATABASE!**
11. Place a KADaoTable component on the form
12. Set Database property to the name of KADaoDatabase component *(by default this is KADaoDatabase1)*
13. Select a table from the TableName property combo box
14. Set Active property to True. **NOW YOU ARE CONNECTED TO THE SELECTED TABLE!**
15. Place a standart Delphi DataSource component on the page
16. Set DataSet property to the name of KADaoTable component *(by default this is* KADaoTable*1)*
17. Place any DBAware component like DBGrid on the form.
18. Set DBGrid's DataSource property to the name of DataSource component *(by default this is DataSource1)*

ALL DONE - YOU ARE NOW IN BUSINESS!

Example:

```
Var
    KADaoDatabase1 : TKADaoDatabase;
    KADaoTable1    : TKADaoTable;
    DataSource1    : TDataSource;
begin
    KADaoDatabase1 := TKADaoDatabase.Create(Self);
    KADaoTable1    := TKADaoTable.Create(Self);
    DataSource1    := TDataSource.Create(Self);;
   KADaoDatabase1.Version='3.6';
    KADaoDatabase1.GoOffline;
    KADaoDatabase1.UserName:='Admin';
    KADaoDatabase1.Password:='SecretWord';
    KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
    KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
    KADaoDatabase1.DatabaseType:='Access';
    KADaoDatabase1.GoOnline;
    KADaoDatabase1.RecreateCore;
    KADaoDatabase1.Database:='d:\demo.mdb';
    KADaoDatabase1.Connected:=True;
    KADaoTable1.Database:=KADaoDatabase1;
    KADaoTable1.TableName:='bnb-1999';
    KADaoTable1.TableType:=dbOpenTable;
    KADaoTable1.Active:=True;
    DataSource1.DataSet:=KADaoTable1;
    DBGrid1.DataSource:=DataSource1;
    ShowMessage('OK You have working application!');
    DataSource1.Free;
    KADaoTable1.Active:=False;
    KADaoDatabase1.Connected:=False;
    KADaoTable1.Free;
    KADaoDatabase1.Free;
end;
```


Object Inspector
KADaoDatabase1: TKADaoDatabase
Properties
Events
ComponentVersion 3.02
Connected False 7
Database d:\demo.mdb 6
DatabaseLanguage
DatabasePassword 4
DatabaseType Access 5
DefaultCursorDriver dbUseDefaultCursor
EngineType dbUseJet
Exclusive False
LoginPrompt False
Name KADaoDatabase1
Password pedal 2
PrivateEngine False
QueryTimeout 60
ReadOnly True
SystemDatabase system.mdb 3
Tag 0
UserName Admin 1
UsesDynaDao True
Version 3.5
VersionDetails 3.51
Workspace DaoWorkspace

How to make an simple application using KADao as a Query component?

1. Place a KADaoDatabase component on the form
2. Set UserName Property
3. Set Password Property
4. If you use different system database then default set SystemDatabase property
5. If your entire database is password protected set DatabasePassword property
6. If your Database is not an MS Access database select apropriate database type from DatabaseType property
7. Choose Database filename or DSN using the Database property.
8. If you want to use an Access XX-97 database set **Version** property to **3.5**
9. If you want to use an Access 2000-2002 database set **Version** property to **3.6**
10. Set Connected property to True. **NOW YOU ARE CONNECTED TO THE DATABASE!**
11. Place a KADaoTable component on the form
12. Set Database property to the name of KADaoDatabase component *(by default this is KADaoDatabase1)*
13. Set SQL property to some text like
    select * from [MyTableName];
14. Set Active property to True. **NOW YOU ARE CONNECTED TO RESULTSET GENERATED BY YOUR QUERY!**
15. Place a standart Delphi DataSource component on the page
16. Set DataSet property to the name of KADaoTable component *(by default this is* KADaoTable*1)*
17. Place any DBAware component like DBGrid on the form.
18. Set DBGrid's DataSource property to the name of DataSource component *(by default this is DataSource1)*

ALL DONE - YOU ARE NOW IN BUSINESS!

Example:

```
Var
    KADaoDatabase1 : TKADaoDatabase;
    KADaoTable1    : TKADaoTable;
    DataSource1    : TDataSource;
begin
    KADaoDatabase1 := TKADaoDatabase.Create(Self);
    KADaoTable1    := TKADaoTable.Create(Self);
    DataSource1    := TDataSource.Create(Self);
   KADaoDatabase1.Version='3.6';
    KADaoDatabase1.GoOffline;
    KADaoDatabase1.UserName:='Admin';
    KADaoDatabase1.Password:='SecretWord';
    KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
    KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
    KADaoDatabase1.DatabaseType:='Access';
    KADaoDatabase1.GoOnline;
    KADaoDatabase1.RecreateCore;
    KADaoDatabase1.Database:='d:\demo.mdb';
    KADaoDatabase1.Connected:=True;
    KADaoTable1.Database:=KADaoDatabase1;
    KADaoTable1.TableName:='SELECT * FROM [bnb-1999];';
    KADaoTable1.TableType:=dbOpenDynaset;
    KADaoTable1.Active:=True;
    DataSource1.DataSet:=KADaoTable1;
    DBGrid1.DataSource:=DataSource1;
    ShowMessage('OK You have working application!');
    DataSource1.Free;
    KADaoTable1.Active:=False;
    KADaoDatabase1.Connected:=False;
    KADaoTable1.Free;
```

```
        KADaoDatabase1.Free;
end;
```

How to make an simple application using KADao as a QueryDEF component?

1. Place a KADaoDatabase component on the form
2. Set UserName Property
3. Set Password Property
4. If you use different system database then default set SystemDatabase property
5. If your entire database is password protected set DatabasePassword property
6. If your Database is not an MS Access database select apropriate database type from DatabaseType property
7. Choose Database filename or DSN using the Database property.
8. If you want to use an Access XX-97 database set **Version** property to **3.5**
9. If you want to use an Access 2000-2002 database set **Version** property to **3.6**
10. Set Connected property to True. **NOW YOU ARE CONNECTED TO THE DATABASE!**
11. Place a KADaoTable component on the form
12. Set Database property to the name of KADaoDatabase component *(by default this is KADaoDatabase1)*
13. Select a QueryDef from the QueryDefName property combo box
14. If your QueryDef requires parameters set them in QueryDefParameters property
15. Set Active property to True. **NOW YOU ARE CONNECTED TO RESULTSET GENERATED BY QUERYDEF!**
16. Place a standart Delphi DataSource component on the page
17. Set DataSet property to the name of KADaoTable component *(by default this is* KADaoTable*1)*
18. Place any DBAware component like DBGrid on the form.
19. Set DBGrid's DataSource property to the name of DataSource component *(by default this is DataSource1)*

ALL DONE - YOU ARE NOW IN BUSINESS!

Example:

```
Var
    KADaoDatabase1 : TKADaoDatabase;
    KADaoTable1    : TKADaoTable;
    DataSource1    : TDataSource;
begin
    KADaoDatabase1 := TKADaoDatabase.Create(Self);
    KADaoTable1    := TKADaoTable.Create(Self);
    DataSource1    := TDataSource.Create(Self);;
   KADaoDatabase1.Version='3.6';
   KADaoDatabase1.GoOffline;
   KADaoDatabase1.UserName:='Admin';
   KADaoDatabase1.Password:='SecretWord';
   KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
   KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
   KADaoDatabase1.DatabaseType:='Access';
   KADaoDatabase1.GoOnline;
   KADaoDatabase1.RecreateCore;
   KADaoDatabase1.Database:='d:\demo.mdb';
   KADaoDatabase1.Connected:=True;
   KADaoTable1.Database:=KADaoDatabase1;
   KADaoTable1.TableName:='MyQueryDef';
      KADaoTable1.QueryDefParameters.Clear;
      KADaoTable1.QueryDefParameters.Add('123');
   KADaoTable1.TableType:=dbOpenDynaset;
   KADaoTable1.Active:=True;
   DataSource1.DataSet:=KADaoTable1;
   DBGrid1.DataSource:=DataSource1;
   ShowMessage('OK You have working application!');
   DataSource1.Free;
```

```
        KADaoTable1.Active:=False;
        KADaoDatabase1.Connected:=False;
        KADaoTable1.Free;
        KADaoDatabase1.Free;
end;
```

How to create an Microsoft Access Database?

The preferred way to create Microsoft Access Database is to use CreateAccessDatabaseEx2 routine. But you can use one of the folowing methods of KADaoDatabase:

1. CreateAccessDatabase
   Example:
   ```
   KADaoDatabase1.CreateAccessDatabase('d:\MyDir\runtimex.mdb');
   ```
2. CreateAccessDatabaseEx
   Example:
   ```
   KADaoDatabase1.CreateAccessDatabaseEx('d:\MyDir\
   runtimex.mdb','0x0409','1252','0','','40',False);
   ```
3. CreateAccessDatabaseEx2
   Example:
   ```
   KADaoDatabase1.CreateAccessDatabaseEx2('d:\MyDir\
   runtimex.mdb',dbLangGeneral,'','40',False);
   ```
   Supported language constants are:
   ```
   dbLangArabic;
   dbLangCzech;
   dbLangDutch;
   dbLangGeneral;
   dbLangGreek;
   dbLangHebrew;
   dbLangHungarian;
   dbLangIcelandic';
   dbLangNordic;
   dbLangNorwDan;
   dbLangPolish;
   dbLangCyrillic;
   dbLangSpanish;
   dbLangSwedFin;
   dbLangTurkish;
   dbLangJapanese;
   dbLangChineseSimplified;
   dbLangChineseTraditional;
   dbLangKorean;
   dbLangThai;
   ```
   Note:
   You can found various language definitions in DaoApi.pas file
   See descriptions of the used methods in Reference section

Example 1:

```
Var
    KADaoDatabase1 : TKADaoDatabase;
begin
    KADaoDatabase1 := TKADaoDatabase.Create(Self);
    KADaoDatabase1.GoOffline;
    KADaoDatabase1.UserName:='Admin';
    KADaoDatabase1.Password:='SecretWord';
    KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
    KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
    KADaoDatabase1.DatabaseType:='Access';
    KADaoDatabase1.GoOnline;
    KADaoDatabase1.RecreateCore;
    KADaoDatabase1.CreateAccessDatabase('d:\MyDir\runtimex.mdb');
    Message('OK You create an database!');
    KADaoDatabase1.Free;
end;
```

Example 2:

```
Var
    KADaoDatabase1 : TKADaoDatabase;
begin
```

```
        KADaoDatabase1 := TKADaoDatabase.Create(Self);
        KADaoDatabase1.UserName:='Admin';
        KADaoDatabase1.Password:='SecretWord';
        KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
        KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
        KADaoDatabase1.DatabaseType:='Access';
        KADaoDatabase1.CreateAccessDatabaseEx('d:\MyDir\
   runtimex.mdb','0x0409','1252','0','','30',False);
        Message('OK You create an database!');
        KADaoDatabase1.Free;
end;
```

Example 3:

```
    Var
        KADaoDatabase1 : TKADaoDatabase;
    begin
        KADaoDatabase1 := TKADaoDatabase.Create(Self);
        KADaoDatabase1.UserName:='Admin';
        KADaoDatabase1.Password:='SecretWord';
        KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
        KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
        KADaoDatabase1.DatabaseType:='Access';
        KADaoDatabase1.CreateAccessDatabaseEx2('d:\MyDir\
   runtimex.mdb',dbLangGeneral,'','30',False);
        Message('OK You create an database!');
        KADaoDatabase1.Free;
end;
```

How to create an Access Table?

Use TKADaoTableManager class to create tables.
Creating a table is very similar to Delphi's TTable.
Use value of 0 for fields with fixed size - Integers for example.
Example:

```
Var
 TM : TKADaoTableManager;
begin
  KADaoDatabase1.Database:='runtime.mdb';
  KADaoDatabase1.Connected  := True;
  TM := TKADaoTableManager.Create(KADaoDatabase1);
  TM.TableName:='Table1';
  TM.FieldDefs.Add('Field 1',ftInteger,0,False);
  TM.FieldDefs.Add('Field 2',ftString,100,False);
  TM.FieldDefs.Add('Field 3',ftDate,0,False);
  TM.IndexDefs.Add('Field 1','Field 1',[ixPrimary,ixUnique]);
  TM.IndexDefs.Add('Field 2','Field 2',[]);
  TM.CreateTable;
  TM.Free;
  KADaoDatabase1.Connected  := False;
End;
```

How to create a Paradox Table?

Use TKADaoTableManager class to create tables.
Creating a table is very similar to Delphi's TTable.
There are some restrictioms on Creating Indexes on Paradox Tables:

BORLAND RESTRICTIONS WHICH APPLY HERE:

- **First field must be Primary Index**
- **Unique Indexes can be created only using Paradox 7.X ISAM driver wich is available only with DAO 3.6**
- **All fields that are in PrimaryKey index must follow the first field**

Example:

```
Var
 TM : TKADaoTableManager;
begin
  KADaoDatabase1.Version='3.6';
  KADaoDatabase1.DatabaseType='Paradox 7.X';
  KADaoDatabase1.Database:='D:\Paradox\MyDb';
  KADaoDatabase1.Connected  := True;
  TM := TKADaoTableManager.Create(KADaoDatabase1);
  TM.TableName:='Table1';
  TM.FieldDefs.Add('Field 1',ftInteger,0,False);
  TM.FieldDefs.Add('Field 2',ftString,100,False);
  TM.FieldDefs.Add('Field 3',ftDate,0,False);
  TM.IndexDefs.Add('Field 1','Field 1',[ixPrimary,ixUnique]);
  TM.IndexDefs.Add('Field 2','Field 2',[]);
  TM.CreateTable;
  TM.Free;
  KADaoDatabase1.Connected  := False;
End;
```

How to use TBlobField?

It is important to know how to Use TBlobFields with KADao

Example (Saving blob field to file):

```
Var
  TBF : TBlobField;
begin
  KAdaoTable1.Active:=True;
  TBF:=KAdaoTable1.FieldByName('FieldName') As TBlobField;
  KAdaoTable1.First;
  TBF.SaveToFile('blobtest.txt');
  KAdaoTable1.Active:=False;
end;
```

Example (Loading blob field from file):

```
Var
  TBF : TBlobField;
begin
  KAdaoTable1.Active:=True;
  TBF:=KAdaoTable1.FieldByName('FieldName') As TBlobField;
  KAdaoTable1.First;
  KAdaoTable1.Edit;
  TBF.LoadFromFile('blobtest.txt');
  KAdaoTable1.Post;
  KAdaoTable1.Active:=False;
end;
```

How to use TBlobStream?

It is also easy to implemets TblobStream functions for reading and writing to files.
See also Description of **CreateBlobStream** method
Examples:

```
SAVING BLOB TO FILE
*****************************************************************************
****
Var
  BS : TBlobStream;
  MS : TMemoryStream;
begin
  KADaoTable1.First;
  MS := TMemoryStream.Create;
  BS :=
TBlobStream(KADaoTable1.CreateBlobStream(KADaoTable1.FieldByName('MyBlobField'),
bmRead));
  MS.LoadFromStream(BS);
  MS.Position:=0;
  MS.SaveToFile('d:\BlobTest.txt');
  MS.Free;
  BS.Free;
end;
LOADING BLOB FROM FILE
*****************************************************************************
*
Var
  BS : TBlobStream;
  MS : TMemoryStream;
begin
  KADaoTable1.First;
  MS := TMemoryStream.Create;
  BS :=
TBlobStream(KADaoTable1.CreateBlobStream(KADaoTable1.FieldByName('MyBlobField'),
bmWrite));
  MS.LoadFromFile('d:\BlobTest.txt');
  MS.Position:=0;
  KADaoTable1.Edit;
  MS.SaveToStream(BS);
  KADaoTable1.Post;
  MS.Free;
  BS.Free;
end;
```

Supported Field types when using KADaoTableManager:

Following field types are supported when using KADaoTableManager
Also you can see to what DAO types they are converted:

```
ftString        --> dbText;
ftSmallint      --> dbInteger;
ftInteger       --> dbLong;
ftWord          --> dbLong;
ftBoolean       --> dbBoolean;
ftFloat         --> dbSingle;
ftCurrency      --> dbCurrency;
ftBCD           --> dbCurrency;
ftDate          --> dbDate;
ftTime          --> dbDate;
ftDateTime      --> dbDate;
ftByte          --> dbText;
ftVarBytes      --> dbMemo;
ftAutoInc       --> dbAutoIncInteger;
ftBlob          --> dbLongBinary;
ftMemo          --> dbMemo;
ftGraphic       --> dbLongBinary;
ftFmtMemo       --> dbMemo;
ftParadoxOle    --> dbLongBinary;
ftDBaseOle      --> dbLongBinary;
ftTypedBinary   --> dbLongBinary;
```

Note:

If You want EXACT DAO types use AddFieldsToTable Method or KADaoField from KADaoDeluxe Package

How to speedup KADAO?

There atr several things to do to speedup KADao to the maximum

1. *If possible always use **KADAO35** or **KADAO36** not DYNADAO. Working with COM early binding is faster then trough late binding of DYNADAO. Made the following changes to CommonDirectives.Pas:*

Example:

```
{DEFINE DYNADAO}
{$DEFINE DAO35}
{DEFINE DAO36}
```

OR

```
{DEFINE DYNADAO}
{DEFINE DAO35}
{$DEFINE DAO36}
```

2. *Always put your code between **StartTransaction** and **Commit** methodts*

This minimizes JET DISK IO operations

Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.StartTransaction;
{do all you need ...}
KADaoDatabase1.Commit
KADaoDatabase1.Close;
```

3. *Always use Insert instead of Append to add new records to a Table*
   Insert INCREASES speed DRAMATICALY since Append DECREASES speed DRAMATICALY!!!

4. *When adding large amount of data use seperate table not linked to any dbaware controls*
   Using just DisableControls/EnableControls slows down by 300%
   *Also you can use **BatchMode** property. See **BatchMode** description on how to use them.*

5. *If no changes are expected from user set **ReadOnly** property to True*
   This will skip some large queries on table fields

6. *Set **UseDaoProperties** and **UseCaptions** to False. This will speedup your **queries** by **10000%***
   But adding new records is not so easy as usual!

7. *If your Application does not display Memos in DBGrids set **CacheMemos** property to False.*

8. *If your Application does not display Images in DBGrids set **CacheBlobs** property to False.*

9. *If your table contains lookup fields set **CacheLookups** property to True.*
   *Don't forget to call **RefreshLookups** method when changes are made to lookup table*

10. *Set **UseGetRecNo** to False*

11. *Set **UseRecordCount** to True*

Constructor Create(AOwner : TComponent);

KADaoDatabase component constructor
Creates KADaoDatabase component
To create KADaoDatabase component without placing it on the Form or DataModule use the following code:

```
Var
   KADaoDatabase1 : TKADaoDatabase;
 Begin
   KADaoDatabase1 := TKADaoDatabase.Create(Self);
   {do some stuff ...}
   KADaoDatabase1.Free;
 End;
```

To create KADaoDatabase component in Console application without placing it on the Form or DataModule use the following code:

```
Var
   KADaoDatabase1 : TKADaoDatabase;
 Begin
   KADaoDatabase1 := TKADaoDatabase.Create(Nil);
   {do some stuff ...}
   KADaoDatabase1.Free;
 End;
```

Destructor Destroy; override;

KADaoDatabase component destructor
Do not use to destroy a KADaoDatabase component
Use Free method instead
Example:

```
KADaoDatabase1.Free;
```

The code supposes that KADaoDatabase1 is variable of type TKADaoDatabase;

Property CoreDBEngine : OleVariant; (With DYNADAO defined)
Property CoreDBEngine : DbEngine; (With DYNADAO undefined)

This is the real DAO object which made all the stuff in working with Microsoft Jet Engine.
You can use it to obtain various information or to execute various methods.
For more information see DAO Help.
Example:

```
Var
   KADaoDatabase1  : TKADaoDatabase;
   NumberOfErrors  :  Integer;
 Begin
   KADaoDatabase1 := TKADaoDatabase.Create(Self);
   NumberOfErrors := KADaoDatabase1.CoreDBEngine.Errors.Count;
   KADaoDatabase1.Free;
 End;
```

Property CoreWorkspace : OleVariant;

This is the real DAO object which made many different things
For more information see DAO Help.
Example:

```
Var
   KADaoDatabase1   : TKADaoDatabase;
   NumberOfUsers    :  Integer;
 Begin
   KADaoDatabase1 := TKADaoDatabase.Create(Self);
   NumberOfUsers  := KADaoDatabase1.CoreWorkspace.Users.Count;
   KADaoDatabase1.Free;
 End;
```

Property CoreDatabase : OleVariant; (With DYNADAO defined)
Property CoreDatabase : Database; (With DYNADAO undefined)

This is the real DAO object which made Database IO.
You can use it to obtain various information or to execute various methods.
For more information see DAO Help.
Example:

```
Var
   KADaoDatabase1  : TKADaoDatabase;
   NumberOfTables  :  Integer;
 Begin
   KADaoDatabase1 := TKADaoDatabase.Create(Self);
   {perform some code to connect to a database}
   {...}
   NumberOfTables := KADaoDatabase1.CoreDatabase.TableDefs.Count;
   KADaoDatabase1.Free;
 End;
```

Property DatabaseLanguageInt : Integer;

Retuns the Language code of the current database.
Language Codes are described in Microsoft DAO And also in DAOApi.pas
Example:

```
Var
   KADaoDatabase1   : TKADaoDatabase;
   Begin
       KADaoDatabase1 := TKADaoDatabase.Create(Self);
       KADaoDatabase1.Database:='Demo.mdb';
       KADaoDatabase1.Connected:=True;
       ShowMessage(IntToStr(KADaoDatabase1.DatabaseLanguageInt));
       KADaoDatabase1.Connected:=False;
        KADaoDatabase1.Free;
   End;
```

Property DatabaseLanguage : String;

Retuns the Language name of the current database based on Language Codes.
Language Codes are described in Microsoft DAO And also in DAOApi.pas
Example:

```
Var
   KADaoDatabase1   : TKADaoDatabase;
   Begin
       KADaoDatabase1 := TKADaoDatabase.Create(Self);
       KADaoDatabase1.Database:='Demo.mdb';
       KADaoDatabase1.Connected:=True;
       ShowMessage(KADaoDatabase1.DatabaseLanguage);
       KADaoDatabase1.Connected:=False;
        KADaoDatabase1.Free;
   End;
```

Property ActiveTableNames: TStringList;

Contain list of ALL currently active KADaoTable components.
You can use them to view how many active tables do you have at any time
Example:

```
Var
   NumberOfOpenTables  :  Integer;
   X                   :  Integer;
 Begin
   NumberOfOpenTables  := KADaoDatabase1.ActiveTableNames.Count;
   For X :=0 to NumberOfOpenTables-1 do
     Begin
       ShowMessage(NumberOfOpenTables.Strings[X]);
     End;
 End;
```

Property DatabaseTypes : TStringList;

Contain list of all database types supported by your DAO version installed
This list is retrieved from Windows registry
Example:

```
Var
   NumberOfDatabaseTypes :  Integer;
   X                     :  Integer;
 Begin
   NumberOfDatabaseTypes := KADaoDatabase1.DatabaseTypes.Count;
   For X :=0 to NumberOfDatabaseTypes-1 do
     Begin
       ShowMessage(DatabaseTypes.Strings[X]);
     End;
 End;
```

Property DSNFileNames : TStringList;

Contain FileNames of the DSN's which are disk based
Format is **DSN=FullPathFileName**
DSN's to database servers are not included in this list
Example:

```
Var
   NumberOfDSNFileNames :  Integer;
   X                    :  Integer;
 Begin
   NumberOfDSNFileNames := KADaoDatabase1.DSNFileNames.Count;
   For X :=0 to NumberOfDSNFileNames-1 do
     Begin
       ShowMessage(DSNFileNames.Strings[X]);
     End;
 End;
```

Property Params : TStringList;

Contains database connection parameters for KADaoDatabase
Params is a list of string items, each representing a different database connection parameter.
Supported parameters are:

Username
Password
DatabasePassword

You can set them using the predefined constants

SzUsername
SzPassword
SzDbPassword

Example:

```
KADaoDatabase1.Params.Clear
KADaoDatabase1.Params.Add(SzUSERNAME+'='+myUsername);
KADaoDatabase1.Params.Add(SzPASSWORD+'='+myPassword);
KADaoDatabase1.Params.Add(SzDBPASSWORD+'='+myDatabasePassword);
KADaoDatabase1.Open;
```

Note:

*If you use this property you **must set all 3 parameters***
If some parameters are empty just set them to empty strings:

```
KADaoDatabase1.Params.Clear
KADaoDatabase1.Params.Add(SzUSERNAME+'='+myUsername);
KADaoDatabase1.Params.Add(SzPASSWORD+'='+myPassword);
KADaoDatabase1.Params.Add(SzDBPASSWORD+'='+'');
```

Property QueryDefNames : TStringList;

Contains list of all QueryDef (Queryes stored in mdb file) names available in the current open database
Example:

```
Var
   NumberOfQueryDefs    :  Integer;
   X                    :  Integer;
 Begin
   NumberOfQueryDefs    := KADaoDatabase1.QueryDefNames.Count;
   For X :=0 to NumberOfQueryDefs-1 do
     Begin
       ShowMessage(QueryDefNames.Strings[X]);
     End;
 End;
```

Property TableNames : TStringList;

Contains list of all Table names available in the current open database
Example:

```
Var
   NumberOfTables      :  Integer;
   X                   :  Integer;
 Begin
   NumberOfTables := KADaoDatabase1.TableNames.Count;
   For X :=0 to NumberOfTables-1 do
     Begin
       ShowMessage(TableNames.Strings[X]);
     End;
 End;
```

Property ComponentVersion : String;

This is a readonly property giving information about the version of KADaoDatabase component
This property is for information purpouses only
Example:

```
If KADaoDatabase1.ComponentVersion='3.0' then ShowMessage('BUGGY VERSION – USE
3.01');
```

Property Connected : Boolean;

False by default;
Connects KADaoDatabase component to a Database
Set to True to open a Database
Set to False to close a Database
Also you can use the **Open** method to do the same.
Example:

```
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
KADaoDatabase1.Database:='db2.mdb';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property Database : String;

Based on DatabaseType property this property contains one of the following

1. The full or relative path and file name to your database if all tables are stored in one file (like Microsoft Access mdb)
   Also can contain only the file name of database.
   Example:

```
KADaoDatabase1.Database:='C:\Program Files\MyAccessFolder\Test.mdb';
KADaoDatabase1.Connected:=True;
```

2. The full path of the folder containing tables if Database is folder based (like Borland Paradox db) **without last backslash**
   Example:

```
KADaoDatabase1.Database:='C:\Program Files\MyParadoxFolder';
KADaoDatabase1.Connected:=True;
```

3. The DSN of an ODBC database if DatabasseType property is set to ODBC
   Example:

```
KADaoDatabase1.Database:='AdvWorks';
KADaoDatabase1.Connected:=True;
```

You can add additional SQL parameters here divided with semicolon
Example:

```
KADaoDatabase1.Database:='AdvWorks;MyParam1;MyParam2;';
KADaoDatabase1.Connected:=True;
```

Property DatabasePassword : String;

Sets password to database protected by password;
This is different from standart Password property since DatabasePassword is one for entire mdb file
Example:

```
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.DatabasePassword := 'SecretWord';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property DatabaseType : String;

Set this parameter to the type of database you want to work with.
Supported databases are different for the DAO 3.5 (Access'97) And DAO 3.6 (Access'2000)
You can get them using **DatabaseTypes** TStringList property
Example:

```
//**************************** Open MS Access database
KADaoDatabase1.DatabaseType:='Access';
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
//**************************** Same as above but using DatabaseTypes property
(Access is always at 0 index)
KADaoDatabase1.DatabaseType:=KADaoDatabase1.DatabaseTypes.Strings[0];
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
//**************************** Open Paradox database
KADaoDatabase1.DatabaseType:='Paradox 5.X';
KADaoDatabase1.Database:='C:\Program Files\MyParadoxFolder';
KADaoDatabase1.Connected:=True;
```

Property DefaultCursorDriver : Integer;

This property is used only when you use and ODBC data source and EngineType property is set to dbUseOdbc
Cursor drivers are different – you can see explanation of each in DAO help
Default is dbUseDefaultCursor
Example:

```
KADaoDatabase1.DatabaseType :='ODBC';
KADaoDatabase1.EngineType   := dbUseOdbc;
KADaoDatabase1.DefaultCursorDriver := dbUseServerCursor;
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property EngineType : Integer;

Sets DBEngine type
When EngineType is set to dbUseOdbc DAO uses ODBCDirect workspaces to work with databases
When EngineType is set to dbUseJet DAO uses Microsoft JET workspaces to work with databases
Example:

```
KADaoDatabase1.DatabaseType :='ODBC';
KADaoDatabase1.EngineType   := dbUseOdbc;
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

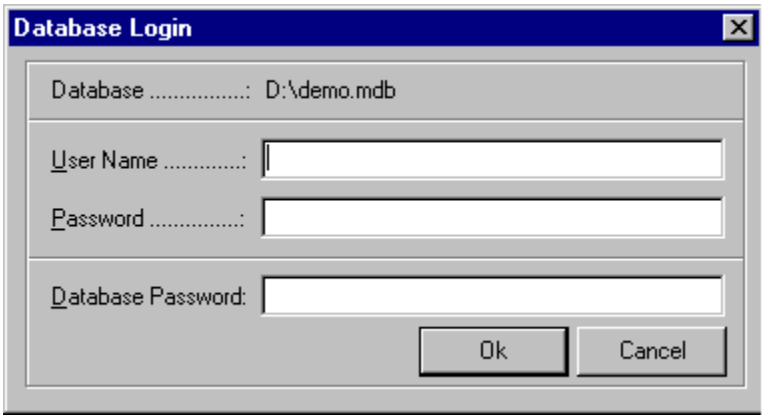
Property Exclusive : Boolean;

When this property is true JET opens Database in Exclusive mode.
This is required for some operations like changing database password.
Example:

```
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.DatabasePassword := 'SecretWord';
KADaoDatabase1.Exclusive:=True;
KADaoDatabase1.Connected:=True;
KADaoDatabase1.ChangeDatabasePassword('SecretWord','NewSecretWord');
KADaoDatabase1.Connected:=False;
KADaoDatabase1.Exclusive:=False;
KADaoDatabase1.DatabasePassword := 'NewSecretWord';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property LoginPrompt : Boolean;

False by default;
Same property as TDatabase.LoginPrompt;
Do not set to True in CGI Scripts and other console applications.
When set to True shows an login dialog window:



Property Password : String;

Contains user's password to logon to a database.

Example:

```
KADaoDatabase1.Database := 'Demo.mdb';
KADaoDatabase1.UserName := 'Admin';
KADaoDatabase1.Password := 'SecretWord';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property PrivateEngine : Boolean;

False by default;
When this property is set to true DAO creates a Private JET Engine
Each Private JET Engine is owned only by process that creates it.
Example:

```
KADaoDatabase1.PrivateEngine := True;
KADaoDatabase1.Database := 'Demo.mdb';
KADaoDatabase1.UserName := 'Admin';
KADaoDatabase1.Password := 'SecretWord';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property QueryTimeout : Integer;

Sets or returns a value that specifies the number of seconds to wait before a timeout error occurs when a query is executed on an ODBC data source. Tt specifies a global value for all queries associated with the database.

Example:

```
KADaoDatabase1.DatabaseType :='ODBC';
KADaoDatabase1.EngineType   := dbUseOdbc;
KADaoDatabase1.QueryTimeout := 90
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property ReadOnly : Boolean;

False by default;
Opens the database in ReadOnly mode;
No changes can be made to the any database propertyes or tables.
KADaoTable checks this property and not allow setting its own ReadOnly property to False when KADaoDatabase's ReadOnly property is true.
Example:

```
KADaoDatabase1.Database := 'Demo.mdb';
KADaoDatabase1.ReadOnly := True;
KADaoDatabase1.UserName := 'Admin';
KADaoDatabase1.Password := 'SecretWord';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property SystemDatabase : String;

The explanation about system database is very long and is based on DAO security mechanism.
So you must first detaily read DAO and Microsoft Access help files and the try to use security
It is important to know the security propertyes setting sequence

1. Set UserName Property
2. Set Password Property
3. Set SystemDatabase property

Example:

```
KADaoDatabase1.UserName := 'Admin';
KADaoDatabase1.Password := 'SecretWord';
KADaoDatabase1.SystemDatabase := 'C:\Security\SysDB.mdw';
KADaoDatabase1.Database := 'Demo.mdb';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property UserName : String;

Sets the Username when logging to a database;
Example:

```
KADaoDatabase1.Database := 'Demo.mdb';
KADaoDatabase1.UserName := 'Admin';
KADaoDatabase1.Password := 'SecretWord';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property UsesDynaDao : Boolean;

A readonly property to determine if Dynadao is used;
Example:

```
If KADaoDatabase1.UsesDynaDao then
   Begin
     KADaoDatabase1.CoreWorkspace := NULL;
   End
Else
   Begin
     KADaoDatabase1.CoreWorkspace := Nil;
   End;
```

Property Version : String; {READONLY when DYNADAO is NOT defined}

A string property to set which of JET engines to use

'3.5' – DAO v. 3.5 and JET v. 3.5 (Access'97)

'3.6' – DAO v. 3.6 and JET v. 4.0 (Access'2000)

Example:

```
If KADaoDatabase1.UsesDynaDao then
   Begin
     KADaoDatabase1.Version := '3.5';
   End;
KADaoDatabase1.Database := 'Demo97.mdb';
KADaoDatabase1.UserName := 'Admin';
KADaoDatabase1.Password := 'SecretWord';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Property VersionDetails : String;

A readonly string property.
Gives informatoin about installed dao version
Can differ from the Version property
Example:

```
KADaoDatabase1.Version := '3.5';
ShowMessage(KADaoDatabase1.VersionDetails); //Will show '3.5' or '3.51';
```

Property Workspace : String;

Sets the name of the workspace DAO objectt
Contains 'DaoWorkspace' by default;
It is important to know the security propertyes setting sequence

1. Set UserName Property
2. Set Password Property
3. Set SystemDatabase property
4. Set Workspace Property

Example:

```
KADaoDatabase1.UserName := 'Admin';
KADaoDatabase1.Password := 'SecretWord';
KADaoDatabase1.SystemDatabase := 'C:\Security\SysDB.mdw';
KADaoDatabase1.Workspace := 'MyWorkspace';
KADaoDatabase1.Database := 'Demo.mdb';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Function GetLastDaoError:TDaoErrRec;

Retuns information about last occured DAO ERROR

The format of TDaoErrRec is:

```
TDaoErrRec=Record
             ErrNo       : Integer;
             Source      : String;
             Description : String;
             HelpFile    : String;
             HelpContext : Integer;
           End;
 PDaoErrRec=^TDaoErrRec;
```

Example:

```
Var
 DaoErr : TdaoErrRec
Begin
   KADaoDatabase1.Database := '::::\Demo.mdb'; //Error in file name
   Try
    KADaoDatabase1.Connected:=True;
   Except
    DaoErr:=KADaoDatabase1.GetLastDaoError;
    ShowMessage(DaoErr.Description);
   End;
```

Function ChangeDatabasePassword(OldPassword,NewPassword:String):Boolean;

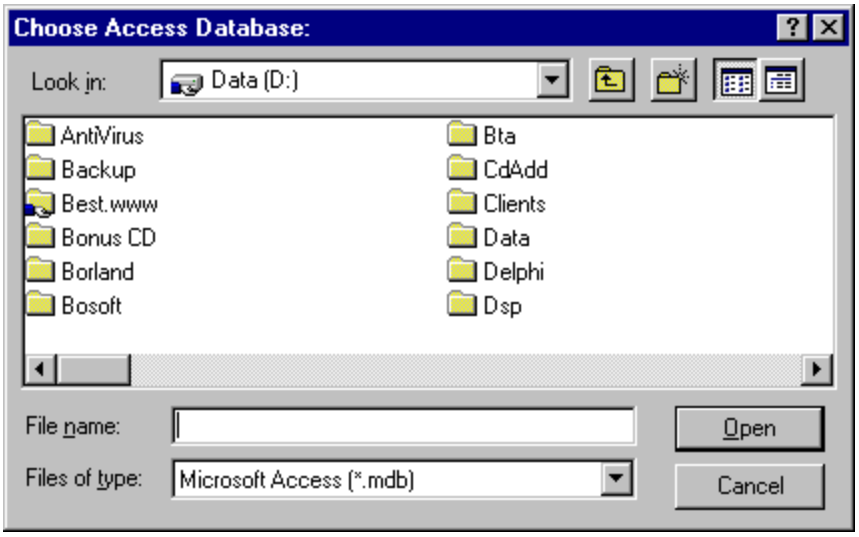
Changes a password on password protected Access database.
OldPassword is the name of the old database password. Set to empty string if database does not have password
NewPassword is the name of the new database password. Set to empty string if you wish to remove database password.

Example:

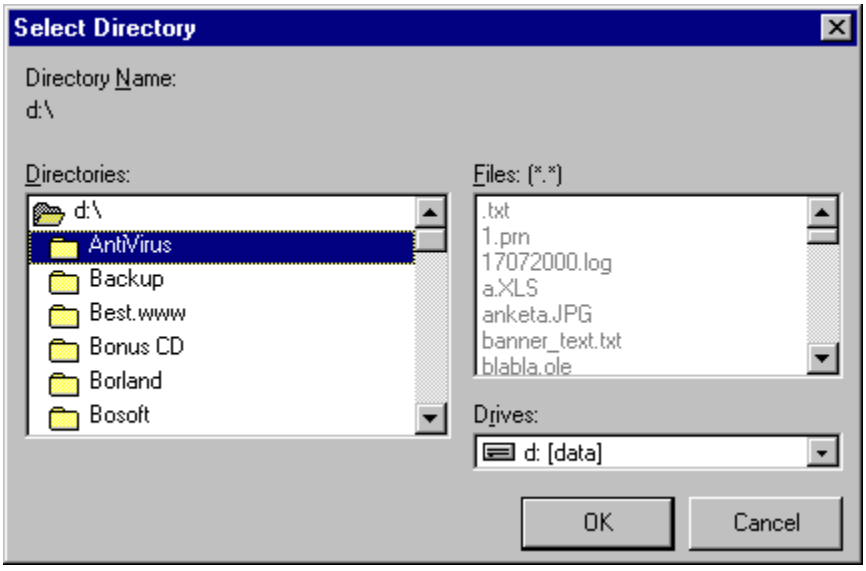
```
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.DatabasePassword := 'SecretWord';
KADaoDatabase1.Exclusive:=True;
KADaoDatabase1.Connected:=True;
KADaoDatabase1.ChangeDatabasePassword('SecretWord','NewSecretWord');

KADaoDatabase1.Connected:=False;
KADaoDatabase1.Exclusive:=False;
KADaoDatabase1.DatabasePassword := 'NewSecretWord';
KADaoDatabase1.Connected:=True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

Function ChooseDatabase: Boolean;

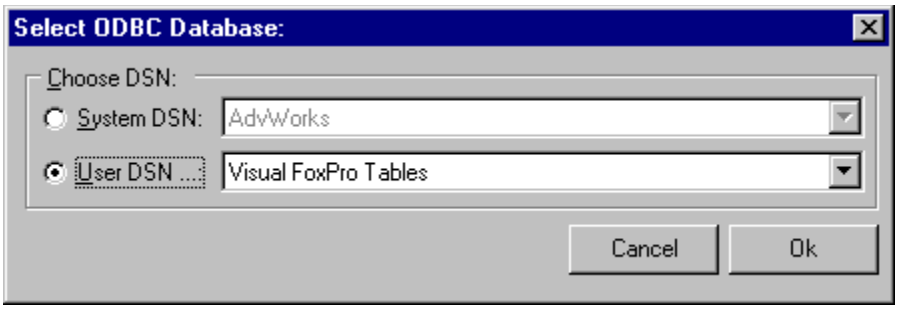
If DatabaseType property is **file based** database displays and OpenFile DialogBox configured for selected DatabaseType;



If DatabaseType property is **folder based** database displays and OpenFile DialogBox configured for selected DatabaseType;



If DatabaseType property is 'ODBC' shows this DialogBox:



If selection succeeds automatically sets Database proprty to selected name;
Example:

```
//**************************** Open MS Access database
KADaoDatabase1.DatabaseType:='Access';
If KADaoDatabase1.ChooseDatabase Then
   Begin
    KADaoDatabase1.Open;
    {do some stuff ...}
    KADaoDatabase1.Close;
   End;
```

Function CreateEmptyTable(TableName:String):Boolean;

Creates an EMPTY table with name **TableName** in currently connected database
Table does not contain any Field and Index definitions
Example:

```
//***************************** Open MS Access database
KADaoDatabase1.DatabaseType:='Access'
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.CreateEmptyTable('MyNewTable');
KADaoDatabase1.Close;
```

Event OnLogin : TLoginEvent;

Same event as Delphi's TDatabase.

Example:

```
procedure TForm1.KADaoDatabase1Login(Database: TKADaoDatabase; LoginParams:
TStrings);
begin
  LoginParams.Values[szUSERNAME]:='Admin';
  LoginParams.Values[szPASSWORD]:='SecretWord';
  LoginParams.Values[szDBPASSWORD]:='DatabaseSecretWord';
End;
```

Function CreateIndex(TableName,FieldName:String;IndexType:Integer):Boolean;

Creates a new index in table **TableName** for field **FieldName**
This function creates index for **only one** field. In KADaoDeluxe Package there is a KADaoIndex component which can create multifield indexes.
IndexType can contain one of the following:

```
1 - Primary Index
2 - Unique Index
4 - Normal Index
```

Example:

```
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.CreateIndex('MyTable','Field1',0);
KADaoDatabase1.Close;
```

Function CreateQueryDef(Name:String;SQL:String):Boolean;

Creates a new QueryDef object stored in the database.
Name is the unique name of the QueryDef; object
SQL contains the Query definition.
If name is empty string a temporary QueryDef object is created. It is not saved in the database.
Example:

```
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.CreateQueryDef('NewQueryDef','Select * from [MyTable];';
KADaoDatabase1.Close;
```

Function CreateTable (TableName:String; FieldNames : Variant; FieldTypes : Variant; FieldSizes : Variant; FieldIndexes:Variant; FieldsRequired:Variant):Boolean;

Variant based metod to create a Table;
Use this metod to create fields with exact DAO Field types. TKADaoTableManager class creates BDE data types which then are convertet to DAO types and for some types conversion result may be not the required by You.
TableName is the name of the new table.
FieldNames is VarArray of String type containing names of the fields.
FieldTypes is VarArray of Integer type containing types of the fields.
FieldSizes is VarArray of Integer type containing sizes of the fields.
FieldIndexes is VarArray of Integer type containing index type of the fields.

Valid values are 0,1, 2 and 4:
0 = NoIndex
1 = Primary index
2 = Unique
4 = NormalIndex

FieldsRequired is VarArray of Integer type containing Required attributes of the field

Valid values are 0 and 1
0 = NotRequired
1 = Required

Creating fields in Paradox table requires some special things to do:

- First field must be Primary Index
- Unique Indexes can be created only using Paradox 7.X ISAM driver wich is available only with DAO 3.6
- All fields that are in PrimaryKey index must follow the first field

Example:

```
Var
  FN,FT,FS,FI,FR:Variant;
 Begin
  FN:=VarArrayCreate([0, 1], varOleStr);
  FT:=VarArrayCreate([0, 1], varInteger);
  FS:=VarArrayCreate([0, 1], varInteger);
  FI:=VarArrayCreate([0, 1], varInteger);
  FR:=VarArrayCreate([0, 1], varInteger);


//**************************************************************************
*******
  FN[0]:='Field 1';    // Name of First Field
  FN[1]:='Field 2';    // Name of Second Field Field

//**************************************************************************
*******
  FT[0]:= dbBoolean;   // FieldType 1 is dbBoolean - see field types in
DaoApi.pas
  FT[1]:= dbText;      // FieldType 2 is dbText - see field types in DaoApi.pas

//**************************************************************************
*******
  FS[0]:=1;            // FieldSize 1 is DefaultSize for Boolean variable
  FS[1]:=111;          // FieldSize 2 is 111 symbols

//**************************************************************************
*******
  FI[0]:=1;            // Field 1 is Indexed No duplicates and primary Index
  FI[1]:=2;            // Field 2 is Indexed, No duplicates

//**************************************************************************
```

```
*******
  FR[0]:=1;            // Field 1 is Required
  FR[1]:=0;            // Field 2 is NOT Required

//*************************************************************************
*******
  // Opens Database and creates the table
  //************************************
  KADaoDatabase1.Database:='runtime.mdb';
  KADaoDatabase1.Connected  := True;
  if KADaoDatabase1.CreateTable('NewTable',FN,FT,FS,FI,FR) Then
ShowMessage('Success using CreateTable!');
  KADaoDatabase1.Connected  := False;
End;
```

Function AddFieldsToTable (TableName:String; FieldNames : Variant; FieldTypes : Variant; FieldSizes : Variant; FieldIndexes:Variant; FieldsRequired:Variant):Boolean;

Variant based metod to add fields to a Table;
Use this metod to create fields with exact DAO Field types. TKADaoTableManager class creates BDE data types which then are convertet to DAO types and for some types conversion result may be not the required by You.
TableName is the name of the new table.
FieldNames is VarArray of String type containing names of the fields.
FieldTypes is VarArray of Integer type containing types of the fields.
FieldSizes is VarArray of Integer type containing sizes of the fields.
FieldIndexes is VarArray of Integer type containing index type of the fields.

Valid values are 0,1, 2 and 4:
0 = NoIndex
1 = Primary index
2 = Unique
4 = NormalIndex

FieldsRequired is VarArray of Integer type containing Required attributes of the field

Valid values are 0 and 1
0 = NotRequired
1 = Required

Creating fields in Paradox table requires some special things to do:

- First field must be Primary Index
- Unique Indexes can be created only using Paradox 7.X ISAM driver wich is available only with DAO 3.6
- All fields that are in PrimaryKey index must follow the first field

Example:

```
Var
  FN,FT,FS,FI,FR:Variant;
 Begin
  FN:=VarArrayCreate([0, 1], varOleStr);
  FT:=VarArrayCreate([0, 1], varInteger);
  FS:=VarArrayCreate([0, 1], varInteger);
  FI:=VarArrayCreate([0, 1], varInteger);
  FR:=VarArrayCreate([0, 1], varInteger);


//**************************************************************************
*******
  FN[0]:='Field 1';    // Name of First Field
  FN[1]:='Field 2';    // Name of Second Field Field

//**************************************************************************
*******
  FT[0]:= dbBoolean;   // FieldType 1 is dbBoolean - see field types in
DaoApi.pas
  FT[1]:= dbText;      // FieldType 2 is dbText - see field types in DaoApi.pas

//**************************************************************************
*******
  FS[0]:=1;            // FieldSize 1 is DefaultSize for Boolean variable
  FS[1]:=111;          // FieldSize 2 is 111 symbols

//**************************************************************************
*******
  FI[0]:=1;            // Field 1 is Indexed No duplicates and primary Index
  FI[1]:=2;            // Field 2 is Indexed, No duplicates

//**************************************************************************
```

```
*******
  FR[0]:=1;            // Field 1 is Required
  FR[1]:=0;            // Field 2 is NOT Required

//*****************************************************************************
*******
  // Opens Database and adds fields to the table
  //*******************************************
  KADaoDatabase1.Database:='runtime.mdb';
  KADaoDatabase1.Connected  := True;
  if KADaoDatabase1.AddFieldsToTable('NewTable',FN,FT,FS,FI,FR) Then
     ShowMessage('Success using AddFieldsToTable!');
  KADaoDatabase1.Connected  := False;
End;
```

Function EmptyTable(TableName:String):Boolean;

This function uses SQL commend to emty an existing Table;
TableName is the name of the Table to empty.
Example:

```
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.Open;
If KADaoDatabase1.EmptyTable('MyTable') Then ShowMessage('Success!');
KADaoDatabase1.Close;
```

Function FindWorkspace(WS:String):Boolean;

This is a utility function
WS is the name of DAO Workspace you wint to find.
It returns True if WS existst in Dao Workspaces collection.
Example:

```
If KADaoDatabase1.FindWorkspace('DaoWorkspace') Then
   Begin

ShowMessage(IntToStr(KADaoDatabase1.CoreDBEngine.Workspaces.Item['DaoWorkspace']
.Users.Count));
   End;
```

Function GetQueryDefSQLText(Name:String):String;

Returns the SQL text of QueryDef object stored in database with name **Name**
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Connected  := True;
ShowMessage(KADaoDatabase1.GetQueryDefSQLText('MyQueryDef'));
KADaoDatabase1.Close;
```

Function HasPrimaryKey(NewTable:OleVariant;PrimaryKeyName:String):Boolean;

Returns information if a passed **PrimaryKeyName** exist in a list of Indexes in table **NewTable**
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Connected  := True;
If KADaoDatabase1.HasPrimaryKey('MyTable','PrimaryKey') Then
   ShowMessage('Table MyTable has Primary Key');
KADaoDatabase1.Close;
```

Procedure RecreateCore;

Call this method to recreate CoreDBEngine and CoreWorkspace
Database must NOT be open - i.e Connected ptroperty is False.

Example 1:

```
Var
    KADaoDatabase1 : TKADaoDatabase;
begin
    KADaoDatabase1 := TKADaoDatabase.Create(Self);
    KADaoDatabase1.GoOffline;
    KADaoDatabase1.UserName:='Admin';
    KADaoDatabase1.Password:='SecretWord';
    KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
    KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
    KADaoDatabase1.DatabaseType:='Access';
    KADaoDatabase1.GoOnline;
    KADaoDatabase1.RecreateCore;
    KADaoDatabase1.CreateAccessDatabase('d:\MyDir\runtimex.mdb');
    Message('OK You create an database!');
    KADaoDatabase1.Free;
end;
```

Function RegisterDatabase(DatabaseName, DriverName:String; Silent:Boolean; Attributes:String):Boolean;

This function creates a new DSN entry into the Windows Registry
It is an interface to the DAO method with the same name.
See Dao help for more information

- **DatabaseName** is the name used in the OpenDatabase method. It refers to a block of descriptive information about the data source. For example, if the data source is an ODBC remote database, it could be the name of the server.
- **DriverName** is the name of ODBC driver
- If **Silent** is False DAO showns ODBC driver dialog boxes that prompt for driver-specific information;
- **Attributes** is a list of keywords to be added to the Windows Registry. The keywords are in a carriage-return–delimited string.

Example:

```
Var
   Attributes : String;
Begin
 Attributes:='Database=pubs'#13#10'Description=Some
Descr'#13#10'OemToAnsi=No'#13#10'Server=Server1';
 KADaoDatabase1.RegisterDatabase "Publishers", "SQL Server", True, Attributes);
End;
```

Procedure Close;

Closes an open Database.
This is the same as setting Connected property to False;
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
{do some stuff ...}
KADaoDatabase1.Close;
```

Procedure CloseDatasets;

Closes all datasets associated with the KADaoDatabase component without disconnecting from the DAO database.
Call CloseDataSets to close all active datasets without disconnecting from the DAO database.
This method is same as Borland's TDatabase.CloseDatasets;
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoTable1.TableName  := 'Table1';
KADaoTable2.TableName  := 'Table2';
KADaoTable2.TableName  := 'Table3';
KADaoTable1.Open;
KADaoTable2.Open;
KADaoTable3.Open;
KADaoDatabase1.CloseDatasets;
```

Procedure CreateAccessDatabase(DatabaseName:String);

Simple method to create Microsoft Access database (mdb);
It assumes the following:

1. Language is set to dbLangGeneral (LangID=409, CodePage=1252);
2. If Version property is 3.5 new Database will be in Microsoft Access'97 format.
3. If Version property is 3.6 new Database will be in Microsoft Access'2000 format.
4. Database is not encrypted
5. Database is not password protected

DatabaseName is the full pathname of the new mdb file.
Example:

```
KADaoDatabase1.CreateAccessDatabase('C:\Program Files\NewDb.mdb');
```

Procedure CreateAccessDatabaseEx(DatabaseName, LANGID, CP, COUNTRY, Password, Version:String; Encrypt:Boolean);

Creates a new Microsoft Access database (mdb);
DatabaseName is the full pathname of the new mdb file.
LANGID is the LanguageID for the new Database
CP is the Code Page for the new Database.
Country is the CountryID for the new Database.
Password is a global database password for the new database. Set to empty string if you wish database to be NOT password protected.
Version is the version of the database ('30' for Microsoft Access'97 format, '40' for Microsoft Access'2000 format)
Encrypt is a boolean variable. When set to true the new database will be encrypted.
The following example creates new database with Neutral language (dbLangGeneral English).
Example:

```
KADaoDatabase1.CreateAccessDatabaseEx('c:\www\
runtimex.mdb','0x0409','1252','0','','40',False);
```

Procedure CreateAccessDatabaseEx2(DatabaseName, Language, Password, Version:String; Encrypt:Boolean);

This is the best way to create new Microsoft Access database.
DatabaseName is the full pathname of the new mdb file.
Language is one of the following constants predefined in DaoApi.pas:

```
dbLangArabic;
dbLangCzech;
dbLangDutch;
dbLangGeneral;
dbLangGreek;
dbLangHebrew;
dbLangHungarian;
dbLangIcelandic';
dbLangNordic;
dbLangNorwDan;
dbLangPolish;
dbLangCyrillic;
dbLangSpanish;
dbLangSwedFin;
dbLangTurkish;
dbLangJapanese;
dbLangChineseSimplified;
dbLangChineseTraditional;
dbLangKorean;
dbLangThai;
```

Password is a global database password for the new database. Set to empty string if you wish database to be NOT password protected.
Version is the version of the database ('30' for Microsoft Access'97 format, '40' for Microsoft Access'2000 format)
Encrypt is a boolean variable. If set to true the new database will be encrypted.
Example:

```
KADaoDatabase1.CreateAccessDatabaseEx2('d:\www\
runtimex.mdb',dbLangGeneral,'','40',False);
```

Procedure CompactAccessDatabase(DatabaseName,Password:String);

This method compacts a Microsoft Access database.
DatabaseName is the full pathname of the mdb file to be compacted;
Password is a global database password if database have one.
If Database does NOT have password set Password to empty string
If Database does NOT have password and Password is not empty string the compacted database will have global password same as Password parameter.
Example:

```
KADaoDatabase1.CompactAccessDatabase('d:\www\runtimex.mdb','');
```

Procedure CompactAccessDatabaseEx(DatabaseName: String; NewLocale : String; Encrypt : Boolean; Decrypt : Boolean; NewVersion : Integer; Password : String);

This is a more detailed and preffered method to compact a Microsoft Access database.
It gives an opportunity to change some database settings.
Also it can be used to change the format of the database to an old version format.
DatabaseName is the full pathname of the mdb file to be compacted;
NewLocale is one of the following constants predefined in DaoApi.pas:

```
dbLangArabic;
dbLangCzech;
dbLangDutch;
dbLangGeneral;
dbLangGreek;
dbLangHebrew;
dbLangHungarian;
dbLangIcelandic';
dbLangNordic;
dbLangNorwDan;
dbLangPolish;
dbLangCyrillic;
dbLangSpanish;
dbLangSwedFin;
dbLangTurkish;
dbLangJapanese;
dbLangChineseSimplified;
dbLangChineseTraditional;
dbLangKorean;
dbLangThai;
```

Encrypt is a boolean variable. If set to true the compacted database will be encrypted.
Decrypt is a boolean variable. If set to true the compacted database will be decrypted.
If both Encrypt and Decrypt are true Database will be Decrypted
NewVersion is the new version of the compacted database.
There are a few predefined versions in DaoApi.pas:

```
dbVersion11;
dbVersion20;
dbVersion30;
dbVersion40;
```

Remember that you cannot create version40 database with Dao35

Password is a global database password if database have one.
If Database does NOT have password set Password to empty string
If Database does NOT have password and Password is not empty string the compacted database will have global password same as Password parameter.
Example:

```
KADaoDatabase1.CompactAccessDatabaseEx('d:\www\
runtimex.mdb','',False,True,0,'');
```

Procedure RepairAccessDatabase(DatabaseName,Password:String);

This method tryes to repair a Microsoft Access database.
DatabaseName is the full pathname of the mdb file to be compacted;
Password is a global database password if database have one.
Note:

In Version 3.6 of DAO RepairDatabase is not supported. It is included in CompactDatabase. So you can use CompactAccessDatabase or CompactAccessDatabaseEx to repair a database. RepairAccessDatabase checks the DAO version and if it is 3.6 calls RepairAccessDatabase mathod;

Example:

```
KADaoDatabase1.RepairAccessDatabase('d:\www\runtimex.mdb','');
```

Procedure DeleteField(TableName,FieldName:String);

Deletes a field with the name **FieldName** in a a table with name **TableName.**
Table must not be open.

This method calls DeleteIndexByFieldName before deleting a field.
See DeleteIndexByFieldName description for more information.

Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.DeleteField('MyTable','MyField');
KADaoDatabase1.Close;
```

Procedure DeleteIndexByFieldName(TableName,FieldName:String);

This method deletes all indexes that contain FieldName as first item in the Index definition.
It does not guarantee that all indexes containing field FieldName will be removed.
For such complicated tasks use KADaoIndex component from KADaoDeluxe package.
Table containing the index must not be open.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.DeleteIndexByFieldName('MyTable','MyField');
KADaoDatabase1.Close;
```

Procedure DeleteIndexByName(TableName,IndexName:String);

This method deletes index with the name **IndexName** from table with name **TableName.**
Table containing the index must not be open.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.DeleteIndexByFieldName('MyTable','MyIndex');
KADaoDatabase1.Close;
```

Procedure DeleteQueryDef(QueryName:String);

This method deletes QueryDef object stored in Database.
QueryName is the name of the QueryDef object.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.DeleteQueryDef('MyQueryDef');
KADaoDatabase1.Close;
```

Procedure DeleteTable(TableName:String);

This method deletes table with name TableName from currently open database.
Table must not be open.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.DeleteTable('MyTable');
KADaoDatabase1.Close;
```

Procedure Idle;

This method calls DAO method Idle.
Sometimes JET does not have enough tome to complete all needed opearations.
So when peforming lenghtly tasks is goot practice to call Idle.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
For X := 0 To 1000 do
    Begin
     KADaoDatabase1.DeleteTable('MyTable'+IntToStr(X));
     Application.ProcessMessages;
     KADaoDatabase1.Idle;
    End;
KADaoDatabase1.Close;
```

Procedure LinkExternalTable(Database,TableName,TableType:String;TableAttributes:Integer);

Links an external table to a Database;
Database is the name of database and depends on TableType parameter.
If for example you want to link a MicrosoftAccess table from another Database ypu must pass something like 'c:\mydir\mymdb.mdb';
TableName is the name of the table from external database.
TableType is one of predefined TableTypes in DaoApi.Pas

```
{ _PredefinedTableTypes}
  dBase_50_Table        = 'dBase 5.0;DATABASE=%s';     {Drive:\Path}
  dBase_III_Table       = 'dBase III;DATABASE=%s';     {Drive:\Path}
  dBase_IV_Table        = 'dBase IV;DATABASE=%s';      {Drive:\Path}
  Excel_30_Table        = 'Excel 3.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_40_Table        = 'Excel 4.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_50_Table        = 'Excel 5.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_80_Table        = 'Excel 8.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Exchange_40_Table     = 'Exchange 4.0;MAPILEVEL=$s'; {Brrrrrrrrrr}
  HTML_Import_Table     = 'HTML Import;DATABASE=%s';   {URL}
  Jet_Table             = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Jet_2x_Table          = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Jet_3x_Table          = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Lotus_WK1_Table       = 'Lotus WK1;DATABASE=%s';     {Drive:\Path\
Filename.wk1}
  Lotus_WK3_Table       = 'Lotus WK3;DATABASE=%s';     {Drive:\Path\
Filename.wk3}
  Lotus_WK4_Table       = 'Lotus WK4;DATABASE=%s';     {Drive:\Path\
Filename.wk4}
  FoxPro_20_Table       = 'FoxPro 2.0;DATABASE=%s';    {Drive:\Path}
  FoxPro_25_Table       = 'FoxPro 2.5;DATABASE=%s';    {Drive:\Path}
  FoxPro_26_Table       = 'FoxPro 2.6;DATABASE=%s';    {Drive:\Path}
  FoxPro_30_Table       = 'FoxPro 3.0;DATABASE=%s';    {Drive:\Path}
  Paradox_3X_Table      = 'Paradox 3.X;DATABASE=%s';   {Drive:\Path}
  Paradox_4X_Table      = 'Paradox 4.X;DATABASE=%s';   {Drive:\Path}
  Paradox_5X_Table      = 'Paradox 5.X;DATABASE=%s';   {Drive:\Path}
  Paradox_7X_Table      = 'Paradox 7.X;DATABASE=%s';   {Drive:\Path}
  Text_Table            = 'Text;DATABASE=%s';          {Drive:\Path}
  ODBC_Table            = 'ODBC;DATABASE=%s;UID=%s;PWD=%s;DSN=%s';
  ODBC_Table_Prompt    =  'ODBC;';
```

TableAttributes is an integer variable which contain various link options used by DAO
It can be a combination of the following constants predefined in DaoApi.pas

```
dbAttachExclusive;
dbAttachSavePWD;
dbSystemObject;
dbAttachedTable;
dbAttachedODBC;
dbHiddenObject;
```

See DAO help for descrptions of these constants.

Example:
{Link an Excel table}

```
  KADaoTable1.Active    := False;
  KADaoTable1.TableName := '';
```

```
  KADaoDatabase1.LinkExternalTable('d:\
ExcelTest.xls','Sheet1$',Excel_80_Table,0);
  KADaoTable1.TableName := 'Sheet1$'; //Remember to add a $ after Excel Table
Name!
  KADaoTable1.Active    := True;
```

Procedure Open;

Opens database specifyed in Database property.
This is equal to:

```
KADaoDatabase1.Connected := True;
```

Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
{do some stuff ...}
KADaoDatabase1.Close;
```

Procedure RefreshDefinitions;

Refresh all Dao collections.
Call this after creating new Tables or QueryDefs using CoreDatabase Object.
Warning! In any case it is not reccomended to use CoreDatabase Object for any purpouses. Use KADaoDatabase methods instead.
This will keep integrity of your data.
Example:

```
Var
  NewTable : OleVariant;
Begin
  KADaoDatabase1.Database:='Demo.mdb';
  KADaoDatabase1.Open;
  NewTable:=OleVariant(KADaoDatabase1.CoreDatabase).CreateTableDef(TableName);
  KADaoDatabase1.CoreDatabase.TableDefs.AppEnd(IDispatch(TVarData(NewTable).vDispa
  tch));
  KADaoDatabase1.RefreshDefinitions;
  KADaoDatabase1.Close;
End;
```

Procedure RefreshLink(Database,TableName,TableType:String);

Updates a link to an external table;
Database is the name of database and depends on TableType parameter.
TableName is the name of the linked table.
TableType is one of predefined TableTypes in DaoApi.Pas

```
{ _PredefinedTableTypes}
  dBase_50_Table        = 'dBase 5.0;DATABASE=%s';     {Drive:\Path}
  dBase_III_Table       = 'dBase III;DATABASE=%s';     {Drive:\Path}
  dBase_IV_Table        = 'dBase IV;DATABASE=%s';      {Drive:\Path}
  Excel_30_Table        = 'Excel 3.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_40_Table        = 'Excel 4.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_50_Table        = 'Excel 5.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_80_Table        = 'Excel 8.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Exchange_40_Table     = 'Exchange 4.0;MAPILEVEL=$s'; {Brrrrrrrrrr}
  HTML_Import_Table     = 'HTML Import;DATABASE=%s';   {URL}
  Jet_Table             = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Jet_2x_Table          = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Jet_3x_Table          = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Lotus_WK1_Table       = 'Lotus WK1;DATABASE=%s';     {Drive:\Path\
Filename.wk1}
  Lotus_WK3_Table       = 'Lotus WK3;DATABASE=%s';     {Drive:\Path\
Filename.wk3}
  Lotus_WK4_Table       = 'Lotus WK4;DATABASE=%s';     {Drive:\Path\
Filename.wk4}
  FoxPro_20_Table       = 'FoxPro 2.0;DATABASE=%s';    {Drive:\Path}
  FoxPro_25_Table       = 'FoxPro 2.5;DATABASE=%s';    {Drive:\Path}
  FoxPro_26_Table       = 'FoxPro 2.6;DATABASE=%s';    {Drive:\Path}
  FoxPro_30_Table       = 'FoxPro 3.0;DATABASE=%s';    {Drive:\Path}
  Paradox_3X_Table      = 'Paradox 3.X;DATABASE=%s';   {Drive:\Path}
  Paradox_4X_Table      = 'Paradox 4.X;DATABASE=%s';   {Drive:\Path}
  Paradox_5X_Table      = 'Paradox 5.X;DATABASE=%s';   {Drive:\Path}
  Paradox_7X_Table      = 'Paradox 7.X;DATABASE=%s';   {Drive:\Path}
  Text_Table            = 'Text;DATABASE=%s';          {Drive:\Path}
  ODBC_Table            = 'ODBC;DATABASE=%s;UID=%s;PWD=%s;DSN=%s';
  ODBC_Table_Prompt    =  'ODBC;';
```

Procedure RenameField(TableName,OldFieldName,NewFieldName:String);

This method renames an Field.
TableName is the name of the table containing the Field.
OldFieldName is the current Field name.
NewFieldName is the new name of the Field.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.RenameField('MyTable','FieldName','NewFieldName');
KADaoDatabase1.Close;
```

Procedure RenameIndex(TableName,OldIndexName,NewIndexName:String);

This method renames an Index.
TableName is the name of the table containing the Index.
OldIndexName is the current Index name.
NewIndexName is the new name of the Index.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.RenameIndex('MyTable','IndexName','NewIndexName');
KADaoDatabase1.Close;
```

Procedure RenameQueryDef(OldQueryName,NewQueryName:String);

This method renames an QueryDef object stored in a database.

Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.RenameQueryDef('QueryDefName','NewQueryDefName');
KADaoDatabase1.Close;
```

Procedure RenameTable(OldTableName,NewTableName:String);

This method renames an Table stored in a database.
OldTableName is the current Table name.
NewTableName is the new name of the Table .
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.RenameTable('MyTableName',''MyNewTableName'');
KADaoDatabase1.Close;
```

Procedure StartTransaction;

Starts a transaction.
All commands to the Database are cashed until Commit or Rollback methods are called.
Some databases does not support transactions.
So calling this method will raise an exception if Database does not support transactions.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.StartTransaction;
{do some stuff ...}
if MessageBox(GetFocus,'Commit all changes now?','Question',MB_IconQuestion or
MB_YesNo)=IDYes then
   KADaoDatabase1.Commit
Else
   KADaoDatabase1.Rollback;
KADaoDatabase1.Close;
```

Procedure Commit;

Commits an transaction.
All changes to the database are stored in database;
Some databases does not support transactions.
So calling this method will raise an exception if Database does not support transactions.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.StartTransaction;
{do some stuff ...}
if MessageBox(GetFocus,'Commit all changes now?','Question',MB_IconQuestion or
MB_YesNo)=IDYes then
   KADaoDatabase1.Commit
Else
   KADaoDatabase1.Rollback;
KADaoDatabase1.Close;
```

Procedure Rollback;

Rollbacks an transaction.
All changes to the database are cancelled;
Some databases does not support transactions.
So calling this method will raise an exception if Database does not support transactions.

Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.StartTransaction;
{do some stuff ...}
if MessageBox(GetFocus,'Commit all changes now?','Question',MB_IconQuestion or
MB_YesNo)=IDYes then
   KADaoDatabase1.Commit
Else
   KADaoDatabase1.Rollback;
KADaoDatabase1.Close;
```

Procedure DBEngineLevel_StartTransaction;

Performs StartTransaction at DBEngine Level.
For Realationship between DBEngine, Workspace and Database objects see DAO Help.

Procedure WorkspaceLevel_StartTransaction;

Performs StartTransaction at Workspace Level.
For Realationship between DBEngine, Workspace and Database objects see DAO Help.

Procedure DBEngineLevel_Commit;

Performs Commit at DBEngine Level.
For Realationship between DBEngine, Workspace and Database objects see DAO Help.

Procedure DBEngineLevel_Rollback;

Performs Rollback at DBEngine Level.
For Realationship between DBEngine, Workspace and Database objects see DAO Help.

Procedure WorkspaceLevel_Commit;

Performs Commit at Workspace Level.
For Realationship between DBEngine, Workspace and Database objects see DAO Help.

Procedure WorkspaceLevel_Rollback;

Performs Rollback at Workspace Level.
For Realationship between DBEngine, Workspace and Database objects see DAO Help.

Constructor Create(AOwner: TComponent);

KADaoTable component constructor
Creates KADaoTable component
To create KADaoTable component without placing it on the Form or DataModule use the following code:

```
Var
   KADaoTable1 : KADaoTable;
 Begin
   KADaoTable1 := TKADaoTable.Create(Self);
   {do some stuff ...}
   KADaoTable1.Free;
 End;
```

To create KADaoTable component in Console application without placing it on the Form or DataModule use the following code:

```
Var
   KADaoTable1 : KADaoTable;
 Begin
   KADaoTable1 := TKADaoTable.Create(Nil);
   {do some stuff ...}
   KADaoTable1.Free;
 End;
```

Destructor Destroy;

KADaoTable component destructor
Do not use to destroy a KADaoTable component
Use Free method instead
Example:

```
KADaoTable1.Free;
```

The code supposes that KADaoTable 1 is variable of type TKADaoTable;

Event OnPostError;

This event occurs when JET Engine tryes to update the data in the Database
Handling this event is same as any TDataset descendant with one difference
If You specify **daFail** in **TdataAction** the table is in unupdated state.
In this case you must call Cancel method in your code.
Example:
Let suppose that we have a table that have unique index on field 'Tekst'

```
//
*************************************************************************
HERE ALL IS OK
KADaoTable1.Append;
KADaoTable1.FieldByName('Tekst').AsString:='11';
KADaoTable1.Post;
//************************************************************ HERE AN
EXCEPTYION IS RAIZED
KADaoTable1.Append;
KADaoTable1.FieldByName('Tekst').AsString:='11'; //here exception is raised
KADaoTable1.Post;
//************************************************************ AND HERE IS
THE HANDLING CODE
procedure TForm1.KADaoTable1PostError(DataSet: TDataSet; E: EDatabaseError;var
Action: TDataAction);
begin
  if KADaoTable1.GetLastDaoError.ErrNo=3022 Then
     Begin
      KADaoTable1.FieldByName('Tekst').AsString:='11K';
      Action:=daRetry;
     End;
end;
```

Note that we can use GetLastDaoError method to review the error type
All error numbers are listed in Microsoft DAO Help.

Property CoreRecordset : OleVariant; {WIDTH DYNADAO DEFINED}
Property CoreRecordset : Recordset; {WIDTH DYNADAO UNDEFINED}

This is the real DAO object which made all the stuff in working with Microsoft Jet Engine Recordsets.
You can use it to obtain various information or to execute various methods.
For more information see DAO Help.
Example:

```
Var
   KADaoTable1     : TKADaoTable;
   NumberOfFields  : Integer;
 Begin
   KADaoTable1             := TKADaoTable.Create(Self);
   KADaoTable1.TableName   := 'Table1';
   KADaoTable1.Open;
   NumberOfFields          := KADaoTable1.CoreRecordset.Fields.Count;
   KADaoTable1.Close;
   KADaoTable1.Free;
 End;
```

Function BookmarkValid(Bookmark: TBookmark): Boolean;

Call BookmarkValid to determine if a specifed bookmark is currently assigned a value. Bookmark specifies the bookmark to test.
BookmarkValid returns True if a bookmark is valid. Otherwise, it returns False.
Note that after calling BookmarkValid current record is cahnged to those pointed by passed Bookmark to BookmarkValid;
Example:

```
Var
 BK : TBookmark;
  Begin
    KADaoTable1.Open;
    KADaoTable1.First;
    BK := KADaoTable1.GetBookmark;
    KADaoTable1.Last;
    if BookmarkValid(BK) Then ShowMessage('Now positioned at the begining of
    table');
    KADaoTable1.Close;
  End;
```

Function CheckFieldsInIndex(KF:TStringList):Boolean;

Check if all field names stored in KF exist in a currently selected Index in the Index property.
Returns True if Index have all fields in its definition.
This is service function. The scope of use of this function is very tight.

Function CompareBookmarks(Bookmark1, Bookmark2: TBookmark): Integer;

Call CompareBookmarks to determine if two bookmarks are identical or not. Bookmark1 and Bookmark2 are the bookmarks to compare.
If the bookmarks differ, CompareBookmarks returns 1. If the Bookmarks are identical, or both bookmarks are nil, CompareBookmarks returns 0.
Example:

```
Var
 BK1 : TBookmark;
 BK2 : TBookmark;
  Begin
    KADaoTable1.Open;
    KADaoTable1.First;
    BK1 := KADaoTable1.GetBookmark;
    KADaoTable1.Last;
    BK2 := KADaoTable1.GetBookmark;
    if CompareBookmarks(BK1,BK2) = 0 Then ShowMessage('Big bug in KADao');
    KADaoTable1.Close;
  End;
```

Function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream;

CreateBlobStream returns a TBlobStream object for reading or writing the data in a specified blob field. Call CreateBlobStream to obtain a stream for reading data from or writing data to a binary large object (BLOB) field. The Field parameter must specify a TBlobField component from the Fields property array. The Mode parameter specifies whether the stream will be used for reading, writing, or updating the contents of the field.

Example:

```
SAVING BLOB TO FILE
*******************************************************************************
****
Var
  BS : TBlobStream;
  MS : TMemoryStream;
begin
  KADaoTable1.First;
  MS := TMemoryStream.Create;
  BS :=
TBlobStream(KADaoTable1.CreateBlobStream(KADaoTable1.FieldByName('MyBlobField'),
bmRead));
  MS.LoadFromStream(BS);
  MS.Position:=0;
  MS.SaveToFile('d:\BlobTest.txt');
  MS.Free;
  BS.Free;
end;
LOADING BLOB FROM FILE
*******************************************************************************
*
Var
  BS : TBlobStream;
  MS : TMemoryStream;
begin
  KADaoTable1.First;
  MS := TMemoryStream.Create;
  BS :=
TBlobStream(KADaoTable1.CreateBlobStream(KADaoTable1.FieldByName('MyBlobField'),
bmWrite));
  MS.LoadFromFile('d:\BlobTest.txt');
  MS.Position:=0;
  KADaoTable1.Edit;
  MS.SaveToStream(BS);
  KADaoTable1.Post;
  MS.Free;
  BS.Free;
end;
```

Function CreateField(FieldName:String;FieldType:Integer;FiledSize:Integer):Boolean;

Creates a new field in the current Table specifyed by TableName Property;
Table must be closed.
For Complex DAO fields use KADaoField component from KADaoDeluxe package.

FieldName is the name of the new field.
FieldType is the type of the new field – see DaoApi pas for predefined field types
FiledSize is the size of the new field in bytes; Set to 0 for known field types
Creating fields in Paradox table requires some special things to do:

- First field must be Primary Index
- Unique Indexes can be created only using Paradox 7.X ISAM driver wich is available only with DAO 3.6
- All fields that are in PrimaryKey index must follow the first field

Use value of 0 for fields with fixed size - Integers for example.

Example:

```
Begin
  KADaoTable1.Close;
  KADaoTable1.CreateField('NewField1',dbText,100);
  KADaoTable1.CreateField('NewField2',dbInteger,0);
  KADaoTable1.Open;
End;
```

Function CreateIndex(FieldName:String;IndexType:Integer):Boolean;

Creates a new index in the current Table specifyed by TableName Property;
Table must be closed.
FieldName is the name of the field that will be indexed
IndexType is the type of the new index;
IndexType can contain one of the following:

```
1 - Primary Index
2 - Unique Index
4 - Normal Index
```

Example:

```
Begin
  KADaoTable1.Close;
  KADaoTable1.CreateIndex('MyField',1);
  KADaoTable1.Open;
 End;
```

Function DeleteField (FieldName:String):Boolean;

Deletes an existing field in the current Table specifyed by TableName Property;
Table must be closed.
Example:

```
Begin
   KADaoTable1.Close;
   KADaoTable1.DeleteField('MyField');
   KADaoTable1.Open;
 End;
```

Function DeleteIndex(FieldName:String):Boolean;

Deletes an existing index in the current Table specifyed by TableName Property;
The index corresponds to the field with name **FieldName.**
Table must be closed.
Example:

```
Begin
   KADaoTable1.Close;
   KADaoTable1.DeleteIndex('MyField');
   KADaoTable1.Open;
 End;
```

Function EmptyTable : Boolean;

This method uses the standard DAO methods to empty the current Table specifyed by TableName Property;
Example:

```
KADaoTable1.TableName  := 'Table1';
KADaoTable1.Open;
If KADaoTable1.EmptyTable Then ShowMessage('Success!');
KADaoTable1.Close;
```

Function ExecSQL(SQL:TStrings):Integer;

Call ExecSQL to execute the INSERT, UPDATE, or DELETE statement contained the SLQ parameter;
Example:

```
Var
 SQL : TStringList;
Begin
  SQL := TStringList.Create;
  SQL.Add('DELETE * FROM MyTable;');
  KADaoTable1.ExecSQL(SQL);
  SQL.Free;
End;
```

Note

The return value specifies the number of affected records.
Also RecordsAffected property is set to the same return value;

Function ExecuteQueryDefSQL:Integer;

Call ExecuteQueryDefSQL to execute the INSERT, UPDATE, or DELETE statement contained the QueryDef object with name stored in **QueryDefName** property.
For SELECT statements, call Open instead of ExecuteQueryDefSQL.
Example:

```
KADaoTable1.QueryDefName   := 'Query1';
KADaoTable1.ExecuteQueryDefSQL;
```

Note

The return value specifies the number of affected records.

Also RecordsAffected property is set to the same return value;

Function ExecuteSQL:Integer;

Call ExecuteSQL to execute the INSERT, UPDATE, or DELETE statement currently assigned to the SQL property.
For SELECT statements, call Open instead of ExecSQL.
Example:

```
Begin
  KADaoTable1.SQL.Clear;
  KADaoTable1.SQL.Add('DELETE * FROM MyTable;');
  KADaoTable1.ExecuteSQL;
End;
```

Note
The return value specifies the number of affected records.
Also RecordsAffected property is set to the same return value;

Function Find_First(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):Boolean;

Find first record where fields with names stored in **KeyFields** contain same values as those stored in **KeyValues** variant array.
Name of fields in **KeyFields** string are semicolon delimited
Returns true if record is found;
Example:

```
KADaoTable1.Find_First('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Next('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Prior('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Last('Company','B',[loPartialKey,loCaseInsensitive]);
```

Function Find_Last(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):Boolean;

Find last record where fields with names stored in **KeyFields** contain same values as those stored in **KeyValues** variant array.
Name of fields in **KeyFields** string are semicolon delimited.
Options are same as Borlands Ttable;
Returns true if record is found;
Example:

```
KADaoTable1.Find_First('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Next('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Prior('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Last('Company','B',[loPartialKey,loCaseInsensitive]);
```

Function Find_Nearest(const KeyValues: array of const):Boolean;

Finds first record whose fields values set by SetKeyFields method match values stored in KeyValues variant array;

Returns true if record is found;

Example:

```
KADaoTable1.SetKeyFields('Company');
       KADaoTable1.Find_Nearest([Edit1.Text]);
```

Function Find_NearestEx(const KeyFields: string; const KeyValues: Variant):Boolean;

Finds first record whose fields specifyed in **KeyFields** match values stored in KeyValues variant array;
Name of fields in **KeyFields** string are semicolon delimited.
Returns true if record is found;
Example:

```
KADaoTable1.Find_NearestEx('Company',Edit1.Text;
```

Function Find_Next(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):Boolean;

Find next record where fields with names stored in **KeyFields** contain same values as those stored in **KeyValues** variant array.
Name of fields in **KeyFields** string are semicolon delimited.
Options are same as Borlands Ttable;
Returns true if record is found;
Example:

```
KADaoTable1.Find_First('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Next('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Prior('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Last('Company','B',[loPartialKey,loCaseInsensitive]);
```

Function Find_Prior(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions):Boolean;

Find previous record where fields with names stored in **KeyFields** contain same values as those stored in **KeyValues** variant array.
Name of fields in KeyFields string are semicolon delimited.
Options are same as Borlands Ttable;
Returns true if record is found;
Example:

```
KADaoTable1.Find_First('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Next('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Prior('Company','B',[loPartialKey,loCaseInsensitive]);
KADaoTable1.Find_Last('Company','B',[loPartialKey,loCaseInsensitive]);
```

Procedure FindNearest(const KeyValues: array of const);

Same as TTable.FindNearest
Uses DAO seek method for fast positioning at record containing values stored in KeyValues array.
Internaly FindNearest calls Seek_NearestEx(KeyValues,'>=');
Example:

```
KADaoTable1.IndexName='MyIndex';
KADaoTable1.FindNearest([Edit2.Text]);
```

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

Same as Ttable.FindKey
Uses DAO seek method for fast positioning at record containing values stored in KeyValues array.
Internaly FindKey calls Seek_NearestEx(KeyValues,'=');
Example:

```
KADaoTable1.IndexName='MyIndex';
KADaoTable1.FindKey([Edit2.Text]);
```

Function GetLastDaoError:TDaoErrRec;

Retuns information about last occured DAO ERROR
Same as KADaoDatabase method.
The format of TDaoErrRec is:

```
TDaoErrRec=Record
             ErrNo       : Integer;
             Source      : String;
             Description : String;
             HelpFile    : String;
             HelpContext : Integer;
           End;
PDaoErrRec=^TDaoErrRec;
```

For more information see help on **KADaoDatabase.GetLastDaoError**

Function GetRows(NumRows:Integer):OleVariant;

Use the GetRows method to copy records from a Recordset. GetRows returns a two-dimensional Variant array. The first subscript identifies the field and the second identifies the row number.
This is a interface to DAO Method GetRows - see DAO help for more information
Function positions current record at the next unread record.
Example:

```
Var
   Rows : OleVariant;
Begin
 KADaoTable1.Open;
 KADaoTable1.First;
 Rows:=KADaoTable1.GetRows(6);
 KADaoTable1.Close;
End;
```

Function GetSourceFieldName(FieldName:String):String;

Returns the source name of the field when field with name **FieldName** is a part of join between two or more tables.

Function GetSourceTableName(FieldName:String):String;

Returns the source name of the Table when field with name **FieldName** is a part of join between two or more tables.

Function Locate(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions): Boolean;

Same method as Table.Locate

Function Lookup(const KeyFields: string; const KeyValues: Variant; const ResultFields: string): Variant;

Same method as Table.Lookup

Function FindFirst: Boolean;
Function FindLast: Boolean;
Function FindNext: Boolean;
Function FindPrior: Boolean;

Same as any TDataset descendant
The only difference is thet before calling one of these functions you must call SetFindData mehod.
Example:

```
Var
   A:Variant;
Begin
       A:=StrToDate('24.4.2000');
       KADaoTable1.SetFindData('Date',A,[]);
       KADaoTable1.FindFirst;
       ShowMessage('FindFirst');
       KADaoTable1.FindNext;
       ShowMessage('FindNext');
       KADaoTable1.FindPrior;
       ShowMessage('FindPrior');
        KADaoTable1.FindLast;
       ShowMessage(FindLast);
End;
```

Procedure SetFindData(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions);

Sets the find information for standard Dataset methods FindFirst, FindNext, FindLast, FindPrior
KeyFields is a list of FieldNames delimited with semicolon
KeyValues is a array of values needed for find
Note: FindFirst, FindNext, FindLast, FindPrior are different then Find_First, Find_Next, Find_Last, Find_Prior
Example:

```
Var
   A:Variant;
Begin
       A:=StrToDate('24.4.2000');
       KADaoTable1.SetFindData('Date',A,[]);
       KADaoTable1.FindFirst;
       ShowMessage('FindFirst');
       KADaoTable1.FindNext;
       ShowMessage('FindNext');
       KADaoTable1.FindPrior;
       ShowMessage('FindPrior');
        KADaoTable1.FindLast;
       ShowMessage(FindLast);
End;
```

Function PercentPosition:Single;

Returns a value of type single indicating the approximate location of the current record in the Table based on a percentage of the records in the Table.

Example:

```
KADaoTable1.TableName  := 'Table1';
KADaoTable1.Open;
KADaoTable1.First;
KADaoTable1.MoveBy(100);
ShowMessage(FloatToStr(KADaoTable1.PercentPosition));
KADaoTable1.Close;
```

Function Requery : Boolean;

Updates the data in a DynasetTable by re-executing the query on which the Table is based.
Query must be a QueryDef object i.e QueryDefName property must be set:
Example:

```
KADaoTable1.QueryDefName := 'Query1';
KADaoTable1.Open;
{do some stufff ...}
{If other users have changed data we calling requery to refresh the data}
{Also if we have set different parameters to QueryDef object we must call
Requery}
KADaoTable1.Requery;
KADaoTable1.Close;
```

Function Seek_Nearest(const KeyValues: array of const):Boolean;

Uses DAO seek method for fast positioning at record containing values stored in KeyValues array.
Values may not be equal
Internaly Seek_Nearest calls Seek_NearestEx(KeyValues,'>=');
Example:

```
KADaoTable1.IndexName='MyIndex';
KADaoTable1.Seek_Nearest([Edit2.Text]);
```

Function Seek_NearestEx(const KeyValues: array of const; SeekType:String):Boolean;

Uses DAO seek method for fast positioning at record containing values stored in KeyValues array. Values may not be equal
An additional parameter is SeekType (String) which can be one of the following: '<', '<=', '=', '>=', '>'
Example:

```
KADaoTable1.IndexName='MyIndex';
KADaoTable1.Seek_Nearest([Edit2.Text],'<=');
```

Procedure GetFieldNames(List: TStrings);

Retrieves all field names in the Table and stores them in passed **List** parameter.
This means all fields not only thouse you set in Field Designer;
Example:

```
Var
  TblList : TStringList;
Begin
  KADaoTable1.Open;
  TblList := TStringList.Create;
  KADaoTable1.GetFieldNames(TblList);
  {do some stuff ...}
  TblList.Free;
  KADaoTable1.Close;
End;
```

Procedure GetIndexNames(List: TStrings);

Retrieves all index names in the Table and stores them in passed **List** parameter.
TableType proberty must be set to StandardTable or SnapshotTable or ForwardOnlyTable;
Example:

```
Var
  IdxList : TStringList;
Begin
  KADaoTable1.TableName  := 'Table1';
  KADaoTable1.TableType  := dbOpenTable;
  KADaoTable1.Open;
  IdxList := TStringList.Create;
  KADaoTable1.GetIndexNames(TblList);
  {do some stuff ...}
  IdxList.Free;
  KADaoTable1.Close;
End;
```

Procedure GotoCurrent(Table: TKADaoTable);

Synchronizes the current record for the Current table with the current record of a specified KADaoTable component.
This works if tables have same KADaoDatabase and same TableName properties.
Example:

```
KADaoTable1.GotoCurrent(KADaoTable2);
```

Procedure SetKeyFields(const KeyFields: string);

Set the key fields needed for Find_Nearest method.

Example:

```
KADaoTable1.SetKeyFields('Company;Customer');
KADaoTable1.Find_Nearest([Edit1.Text,Edit2.Text]);
```

Procedure SetKeyParam(const KeyFields: Array of String;const KeyValues: array of const);

Sets the find information for GotoKey method
KeyFields is a array of strings containing FieldNames to include in search
KeyValues is a array of values needed for searching
Example:

```
KADaoTable1.SetKeyParam(['Date'],['24.4.2000']);
KADaoTable1.GotoKey;
```

Function GotoKey: Boolean;

Same as TTable.GotoKey
You must call SetKey before using GotoKey
Returns true if specifyed record is found, false othervise
Example:

```
KADaoTable1.SetKey;
 KADaoTable1.FieldByName('AUD').AsString := '1062.6';
 KADaoTable1.EditKey;
 KADaoTable1.FieldByName('AUD').AsString := '1073.16';
 KADaoTable1.GotoKey;
```

Procedure LockTable(LockType: TLockType);

Same as Ttable.LockTable
type TLockType = (ltReadLock, ltWriteLock);
After calling CurrentRecord is first record in table
Example:

```
KADaoTable1.LockTable(ltReadLock);
```

Procedure UnlockTable(LockType: TLockType);

Same as Ttable.UnlockTable

type TLockType = (ltReadLock, ltWriteLock);

After calling CurrentRecord is first record in table

Example:

```
KADaoTable1.UnlockTable(ltReadLock);
```

Property Active:Boolean;

If set to True opens a table otherwise closes a table;
Same as Open method.
Example:

```
KADaoTable1.TableName  := 'Table1';
KADaoTable1.TableType  := dbOpenTable;
KADaoTable1.Active := True;
{do some stuff ...}
KADaoTable1.Active := False;
```

Property FieldNames : TStrings;

Contains all field names in the Table.
This means all fields not only thouse you set in Field Designer;
Example:

```
For X := 0 To KADaoTable1.FieldNames.Count-1 do
ShowMessage(KADaoTable1.FieldNames.Strings[X]);
```

Property ComponentVersion: String;

This is a readonly property giving information about the version of KADaoTable component
This property is for information purpouses only
Example:

```
If KADaoTable 1.ComponentVersion='3.0' then ShowMessage('BUGGY VERSION – USE
3.01');
```

Property Database: TKADaoDatabase;

This property must be set to the KADaoDatabase which is connected to Database containing Table you want to open.

Example:

```
KADaoTable1.Database   := KADaoDatabase1;
KADaoTable1.TableName  := 'Table1';
KADaoTable1.TableType  := dbOpenTable;
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property Filter : String;

Set this property to assign a filter to your KadaoTable.
Filter is text like SQL but without where clause.
The Filter is not executed until Filtered property is not set to True;
If Filtered is True and KADaoTable.Filter text is changed the Table refresh its contents regarding to new Filter settings.
Example:

```
KADaoTable1.Database   := KADaoDatabase1;
KADaoTable1.TableName  := 'Table1';
KADaoTable1.Open;
KADaoTable1.Filter     := '[NumericField] > 100 And [TextField] Like "Kiril*"
KADaoTable1.Filtered   := True;
{do some stuff ...}
KADaoTable1.Close;
```

Property Filtered: Boolean;

Activates filtering for as table.
False by default;
Filtering may be set by Filter poperty or by handling the OnFilterRecord event;
If Filtered is True and KADaoTable.Filter text is changed the Table refresh its contents regarding to new Filter settings.
Example:

```
KADaoTable1.Database   := KADaoDatabase1;
KADaoTable1.TableName  := 'Table1';
KADaoTable1.Open;
KADaoTable1.Filter     := '[NumericField] > 100 And [TextField] Like "Kiril*"
KADaoTable1.Filtered   := True;
{do some stuff ...}
KADaoTable1.Close;
```

Property IndexName : String;

Set a Index for a specifyed Table.
Setting the Index property rearanges records in the table.
You can use IndexName only on standard tables.
Example:

```
KADaoTable1.Database   := KADaoDatabase1;
KADaoTable1.TableName  := 'Table1';
KADaoTable1.TableType  := dbOpenTable;
KADaoTable1.IndexName  := 'Field1';
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property IndexFieldCount : Integer;

Returns number of fields that make up the current index for the dataset.

Example:

```
KADaoTable1.Database    := KADaoDatabase1;
KADaoTable1.TableName  := 'Table1';
KADaoTable1.TableType  := dbOpenTable;
KADaoTable1.IndexName  := 'Field1';
KADaoTable1.Open;
ShowMessage(IntToStr(KADaoTable1.IndexFieldCount);
KADaoTable1.Close;
```

Property IndexFieldNames : String;

If you write to this property it works as an alternative method of specifying the index to use for a table.
If you read from this property you will get names of all fields participating in the Index
FieldNames are semicolon delimited

Hint:

When first char is "!" and you write to this property an exact search is executed
Let suppose we have two indexes:
Index1=[Field1,Field2]
Index2=[Field1]
then KADaoTable1.IndexFieldNames:='Field1' will return Index1
but KADaoTable1.IndexFieldNames:='**!Field1**' will return Index2
Example:

```
KADaoTable1.IndexFieldNames:='Field1;Field2;Field3';
ShowMessage(KADaoTable1.IndexName);
ShowMessage(KADaoTable1.IndexFieldNames);
```

Property LockEdits : Boolean;

A ReadOnly property that gives information about lock mode in which the table is opened.
If LockEdits is True then Pessimisting locking is in use and 2K page of table containing the record is locked.
If LockEdits is False then Optimisting locking is in use and 2K page of table containing the record is NOT locked.
Example:

```
KADaoTable1.Database   := KADaoDatabase1;
KADaoTable1.TableName  := 'Table1';
KADaoTable1.Open;
if KADaoTable1.LockEdits Then
   Begin
    ShowMessage('Pessimisting locking in use!');
   End
Else
   Begin
    ShowMessage('Optimisting locking in use!');
   End;
{do some stuff ...}
KADaoTable1.Close;
```

Property LockType: Integer;

Sets the LockType when opening a Table.
It can be one of the following values:

```
dbPessimistic
dbOptimistic
dbOptimisticValue
dbReadOnly
dbOptimisticBatch
```

See DAO help for more information about these constants.
Example:

```
KADaoTable1.Database   := KADaoDatabase1;
KADaoTable1.TableName  := 'Table1';
KADaoTable1.LockType   := dbPessimistic;
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property MasterFields : TStrings;

Same as Borland's TTable.MasterFields property;
Remember that TableType of Detail Table must be DynasetTable;
Example:

```
KADaoDatabase1.Database:='master.mdb';
KADaoDatabase1.Connected:=True;
//****************** Setup Master Table properties
KADaoTable1.Database:=KADaoDatabase1;
KADaoTable1.TableName:='CUSTOMER';
KADaoTable1.Active:=True;
Datasource1.DataSet:=KADaoTable1;
DBGrid1.DataSource:=Datasource1;
//****************** Setup Detail Table properties
KADaoTable2.Database:=KADaoDatabase1;
KADaoTable2.TableName:='ORDERS';
KADaoTable2.TableType:=dbOpenDynaset;
KADaoTable2.MasterSource:=Datasource1;
KADaoTable2.MasterFields.Clear;
KADaoTable2.MasterFields.Add('CustNo -> CustNo');
KADaoTable2.Active:=True;
Datasource2.DataSet:=KADaoTable2;
DBGrid2.DataSource:=Datasource2;
```

Property MasterSource : TDataSource;

Same as Borland's TTable.MasterSource property;
Remember that TableType of Detail Table must be DynasetTable;
Example:

```
KADaoDatabase1.Database:='master.mdb';
KADaoDatabase1.Connected:=True;
//****************** Setup Master Table properties
KADaoTable1.Database:=KADaoDatabase1;
KADaoTable1.TableName:='CUSTOMER';
KADaoTable1.Active:=True;
Datasource1.DataSet:=KADaoTable1;
DBGrid1.DataSource:=Datasource1;
//****************** Setup Detail Table properties
KADaoTable2.Database:=KADaoDatabase1;
KADaoTable2.TableName:='ORDERS';
KADaoTable2.TableType:=dbOpenDynaset;
KADaoTable2.MasterSource:=Datasource1;
KADaoTable2.MasterFields.Clear;
KADaoTable2.MasterFields.Add('CustNo -> CustNo');
KADaoTable2.Active:=True;
Datasource2.DataSet:=KADaoTable2;
DBGrid2.DataSource:=Datasource2;
```

Property OpenOptions : TOOSet;

Set various open options for opening the Database.
All open options are described in DAO help;
This set is empty by default;
If Table is already opened then after calling CurrentRecord is first record in table

```
TOO     = (
            dbDenyWrite,
            dbDenyRead,
            dbReadOnly,
            dbAppendOnly,
            dbInconsistent,
            dbConsistent,
            dbSQLPassThrough,
            dbFailOnError,
            dbForwardOnly,
            dbSeeChanges,
            dbRunAsync,
            dbExecDirect
            );
TOOSet = Set of TOO;
```

Example:

```
KADaoTable1.Database        := KADaoDatabase1;
KADaoTable1.TableName       := 'Table1';
KADaoTable1.OpenOptions     := dbDenyRead OR dbFailOnError;
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property ParamCheck : Boolean;

True by default;
If set to True enables parameter checking for SQL queryes.

Property ParamCount : Word;

Count of parameters for an SQL Query;

Property Params : TParams;

Same as Borlands TQuery.Params;

Property QueryDefName : String;

Name of QueryDef object stored in Database to Open or Execute.
Example:

```
KADaoTable1.Database        := KADaoDatabase1;
KADaoTable1.QueryDefName    := 'Query1';
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property QueryDefODBCMaxRecords : Integer;

Sets the number of records to return from an SQLPassTrough Query;
Not Fully supported by KADao;

Property QueryDefParameters : TStrings;

Parameters to pass to an QueryDef object stored in Database;
If some parameters are of Input/Output type after calling Open method will contain results from QueryDef.
The following example will show all records from table [bnb-1999] where field ID contaits value > 123
Example:

```
{********************************************** SQL text of the QueryDef
object ****}
PARAMETERS ID_ IEEESingle;
SELECT * FROM [bnb-1999] WHERE (([bnb-1999].ID)>ID_) ORDER BY ID;
{********************************************** Pascal code ****}
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.QueryDefName   := 'Query1';
KADaoTable1.QueryDefParameters.Clear;
KADaoTable1.QueryDefParameters.Add('123');
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property QueryDefParametersText : TStrings;

This is the same property as QueryDefParameters but without bringing a QueryDefParametersEditor at design time;
If some parameters are of Input/Output type after calling Open method will contain results from QueryDef.
The following example will show all records from table [bnb-1999] where field ID contaits value > 123
Example:

```
{********************************************** SQL text of the QueryDef
object ****}
PARAMETERS ID_ IEEESingle;
SELECT * FROM [bnb-1999] WHERE (([bnb-1999].ID)>ID_) ORDER BY ID;
{********************************************** Pascal code ****}
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.QueryDefName   := 'Query1';
KADaoTable1.QueryDefParametersText.Clear;
KADaoTable1.QueryDefParametersText.Add('123');
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property QueryDefSQLText : TStrings;

Return the SQL text definition of the QueryDef object stored in the Database;
Example:

```
Var
 X:Integer;
Begin
   KADaoTable1.Database       := KADaoDatabase1;
   KADaoTable1.QueryDefName   := 'Query1';
   For X := 0 to QueryDefSQLText.Count-1 do
       ShowMessage(QueryDefSQLText.Strings[X]);

   {********************************************** Will show the following ****}
   PARAMETERS ID_ IEEESingle;
   SELECT * FROM [bnb-1999] WHERE (([bnb-1999].ID)>ID_) ORDER BY ID;
   {*****************************************************************************
   }
End;
```

Property QueryDefType : String;

Returns the type of the QueryDef object as text;
Can be one of the following:

```
dbQSelect;
dbQProcedure;
dbQAction;
dbQCrosstab;
dbQDelete;
dbQUpdate;
dbQAppend;
dbQMakeTable;
dbQDDL;
dbQSQLPassThrough;
dbQSetOperation;
dbQSPTBulk;
dbQCompound;
```

Example:

```
KADaoTable1.Database        := KADaoDatabase1;
KADaoTable1.QueryDefName    := 'Query1';
ShowMessage(KADaoTable1.QueryDefType);
```

See predefined constants in DaoApi.pas;

Property QueryDefTypeInt : Integer;

Same as QueryDefType but returns Integer constant.
Can be one of the following:

```
dbQSelect;
dbQProcedure;
dbQAction;
dbQCrosstab;
dbQDelete;
dbQUpdate;
dbQAppend;
dbQMakeTable;
dbQDDL;
dbQSQLPassThrough;
dbQSetOperation;
dbQSPTBulk;
dbQCompound;
```

Example:

```
KADaoTable1.Database        := KADaoDatabase1;
KADaoTable1.QueryDefName    := 'Query1';
ShowMessage(KADaoTable1.QueryDefType);
```

See predefined constants in DaoApi.pas;

Property QueryDefReturnParams : OleVariant;

Contains an VarArray of return parameters from QueryDefObject if it returns data to caller;
Example:

```
Var
 ReturnParams : OleVariant;
Begin
    KADaoTable1.Database       := KADaoDatabase1;
    KADaoTable1.QueryDefName   := 'Query1';
    KADaoTable1.QueryDefParametersText.Clear;
    KADaoTable1.QueryDefParametersText.Add('123');
    KADaoTable1.Open;
    ReturnParams:=KADaoTable1.QueryDefReturnParams;
    KADaoTable1.Close;
 End;
```

Property ReadOnly: Boolean;

False by default;
When set to True opens the Table in ReadOnly mode;
If ReadOnly is set to False and KADaoDatabase.ReadOnly is True then when calling Open method ReadOnly property is set to True;

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.ReadOnly       := True;
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property SortedBy: TStrings;

Defines sort order of resultset table;
You **cannot** use it with StandardTable type;
The following example opens Table Table1 sorted first by Field **ID** in ascending order and then by Field **Name** in descending order.
Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.TableType      := dbOpenDynaset;
KADaoTable1.Open;
KADaoTable1.SortedBy.Clear;
KADaoTable1.SortedBy.Add('[ID] ASC');
KADaoTable1.SortedBy.Add('[Name] DESC');
KADaoTable1.Sort;
{do some stuff ...}
KADaoTable1.Close;
```

Property SortedByText : TStrings;

This is the same property as QueryDefParameters but without bringing a SortedByEditor at design time;
You **cannot** use it with StandardTable type;
The following example opens Table Table1 sorted first by Field **ID** in ascending order and then by Field **Name** in descending order.
Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.TableType      := dbOpenDynaset;
KADaoTable1.Open;
KADaoTable1.SortedByText.Clear;
KADaoTable1.SortedByText.Add('[ID] ASC');
KADaoTable1.SortedByText.Add('[Name] DESC');
KADaoTable1.Sort;
{do some stuff ...}
KADaoTable1.Close;
```

Property SQL : TStrings;

Contains the SQL text for returning Resultset or Execute query;
Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableType      := dbOpenDynaset;
KADaoTable1.SQL.Clear;
KADaoTable1.SQL.Add('Select * from [Table1]');
KADaoTable1.SQL.Add('Where [Table1.ID] > 100;');
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property TableDateCreated: String;

Returs the Date when Table/QueryDef object is created;

Returns value ONLY when TableType is dbOpenTable

Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.TableType      := dbOpenTable;
KADaoTable1.Open;
ShowMessage(KADaoTable1.TableDateCreated);
KADaoTable1.Close;
```

Property TableLastUpdated: String;

Returs the Date when Table/QueryDef object is last updated;

Returns value ONLY when TableType is dbOpenTable

Example:

```
KADaoTable1.Database        := KADaoDatabase1;
KADaoTable1.TableName       := 'Table1';
KADaoTable1.TableType       := dbOpenTable;
KADaoTable1.Open;
ShowMessage(KADaoTable1.TableLastUpdated);
KADaoTable1.Close;
```

Property TableName : String;

Set this property to the table name you wish to open.
At design time shows a combobox with list of all tables in the selected Database;

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

Property TableType : Integer;

Sets the type of the Recordset object.
Can be one of the following:

```
DbOpenTable=StandardTable
DbOpenDynaset=DynasetTable
DbOpenDynamic=DynamicTable
DbOpenSnapshot=SnapshotTable
DbOpenForwardOnly=ForwardOnlyTable
```

Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.TableType      := DbOpenSnapshot;
KADaoTable1.Open;
{do some stuff ...}
KADaoTable1.Close;
```

See predefined constants in DaoApi.pas;

Property UseBrackets : Boolean;

True by default;
When set to True this property It places Field names in squire brackets "[ ]" when using Locate, Lookup, and Master/Detail
Since squire brackets are MS Access specific turn this property to FALSE when using other databases than MDB

Property RecordsAffected : Integer;

Returns the number of affected records when using one of the following methofs:

ExecSQL
ExecuteQueryDefSQL
ExecuteSQL

Example:

```
KADaoTable1.QueryDefName   := 'Query1';
KADaoTable1.ExecuteQueryDefSQL;
ShowMessage(IntToString(KADaoTable1.RecordsAffected)+' records were changed!');
```

Property SQLExecutionType : Integer;

Set the way on which a query is executed.
By default is dbFailOnError;
Example:

```
KADaoTable1.QueryDefName    := 'Query1';
KADaoTable1.SQLExecutionType := dbFailOnError;
KADaoTable1.ExecuteQueryDefSQL;
```

Event AfterCancel;
Event AfterClose;
Event AfterDelete;
Event AfterEdit;
Event AfterInsert;
Event AfterOpen;
Event AfterPost;
Event AfterScroll;
Event BeforeCancel;
Event BeforeClose;
Event BeforeDelete;
Event BeforeEdit;
Event BeforeInsert;
Event BeforeOpen;
Event BeforePost;
Event BeforeScroll;
Event OnCalcFields;
Event OnDeleteError;
Event OnEditError;
Event OnFilterRecord;
Event OnNewRecord;
Same as any TDataset descendant;

About KADao

KADao is a native DAO component for Delphi/CBuilder
It is the first freeware component to completly access all databases
supported by Microsoft DAO (Data Access Objects) including mdb, xls, dbf etc..
BDE is not required. Microsoft(r) DAO(tm) must be Installed in order component to run.
Support for both Microsoft(r) Access'xx-Access'97 and Microsoft(r) Access'2000
Features:

1. Create, Repair, Compact, Encrypt Access'97 and Access'2000 MDB files
2. Create tables, add indexes, and fields to existing tables and so on.
3. Work as a Table an Query Component supporting both Queries and QueryDefs
4. Compatible with all data aware controls
5. Master/Detail support
6. Locate, Lookup support
7. Find_First, Find_Next, Find_Last, Find_Prior
8. Seek_Nearest, Seek_NearestEx

KADao is a set of tree components to use with Microsoft(r) DAO(tm) Engine
KADao is FREE for commercial and non-commercial use i.e it is absolutely free!
It is tested with Delphi 3.0 And Delphi 5.0 but must work without problems
with any version of Delphi 3-5

Note:
If you are an MS Access proffesional a KADaoDeluxe package may be of interest for you.
KADaoDeluxe is a set of about 20 components which will raise your productivity when working with DAO
Visit my website for more information about KADaoDeluxe

DISCLAIMER OF WARRANTY

COMPONENTS ARE SUPPLIED "AS IS" WITHOUT WARRANTY OF ANY KIND. THE AUTHOR DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR ANY PURPOSE. THE AUTHOR ASSUMES NO LIABILITY FOR DAMAGES, DIRECT OR CONSEQUENTIAL, WHICH MAY RESULT FROM THE USE OF COMPONENTS.
USE THIS COMPONENTS AT YOUR OWN RISK

E-mail : kiril@pari.bg
Website:
www.delphi.pari.bg
or
www.delphiwarrior.freeservers.com

Property Active : Boolean;

Set to true to obtain info about a mdb database

Property Database : String;

Set to full pathname of the Database you want to get info

Property DaoInfoDll : String

By default it points to 'msldbusr.dll' - this is the dll intefacing with DAO

Property DatabaseVersion : Integer;

Returns the version of the selected Database
30 for MS Access'97
40 for MS Access'2000

Property ErrorUsers : TStringList;

Returns the name of machines for users caused crashing of Database

Property LastError : Integer;

Returns the last error when all properties are retrieved

Property LastErrorText : String;

Returns the last error text representation when all properties are retrieved

Property LoggedUsers : TStringList;

Returns the name of machines for all users connected to Database since it is open in share mode

Property LoggedUsersEx : TStringList;

Returns the name of machines/usernames for all users connected to Database since it is open in share mode

Property LoggedNowUsers : TStringList;

Returns the name of machines for currently logged users

Property NumberOfUsers : Integer;

Returns the total number of users logged to database

Function KAGetDatabaseVersion(LibraryName,DatabasePath:String):Integer;

Fills **DatabaseVersion** property

Function KAGetNumberOfUsers(LibraryName,DatabasePath:String):Integer;

Fills **NumberOfUsers** property

Function KAGetLoggedUsers(LibraryName,DatabasePath:String):Integer;

Fills **LoggedUsers** property

Function KAGetLoggedInfo(DatabasePath:String):Boolean;

Fills **LoggedUsersEx** property

Function KAGetLoggedNowUsers(LibraryName,DatabasePath:String):Integer;

Fills **LoggedNowUsers** property

Function KAGetErrorUsers(LibraryName,DatabasePath:String):Integer;

Fills **ErrorUsers** property

Event AfterGetDatabaseVersion

Triggered after filling **DatabaseVersion** property

Event AfterGetNumberOfUsers

Triggered after filling **NumberOfUsers** property

Event AfterGetLoggedUsers

Triggered after filling **LoggedUsers** property

Event AfterGetLoggedUsersEx

Triggered after filling **LoggedUsersEx** property

Event AfterGetLoggedNowUsers

Triggered after filling **LoggedNowUsersEx** property

Event AfterGetErrorUsers

Triggered after filling **ErrorUsers** property

Introduction

KADaoInfo is a small FREE component for using Microsoft's msldbusr.dll to get information about an Access Database without opening it
Also it uses a DOCUMENTED techiques to get usernames of all logged to database users
It can be very usefull in some cases.

Introduction

KADaoDatabase is a main inteface to Microsoft (r) DAO (tm) Database Engine
It can operate with unlimited number of KADaoTable components providing interface to one database

Introduction

KADaoTable is a TDataSet descendant poviding Delphi standard inteface to Microsoft (r) DAO (tm) Database Engine
You can use KADaoTable with any DBAware components wich is main goal of thise components

Microsoft DAO bugs ??? (Unofficial)

The main bug is when appending records to the table opened as StandardTable (dbOpenTable) and a Rollback is executed

If you does not open your table in StartTransaction section the following happens

a.) No records are append to the table (which is good)

b.) RecordCount of the table always increased by number of records that was added (which is VERY BAD)

The following code demontstrates the error:

```
//******************* Here KADaoTable1.RecordCount is 5
KADaoTable1.TableType=dbOpenTable;
KADaoTable1.Open;
KADaoTable1.First;
KADaoDatabase1.StartTransaction;
For X := 1 to 1000 do
   Begin
         KADaoTable1.Insert;
         KADaoTable1.Post;
   End;
KADaoDatabase1.Rollback;
KADaoTable1.Close;
//******************* Here KADaoTable1.RecordCount is 1005 !!!!
```

The CORRECT way to do so is:

```
//******************* Here KADaoTable1.RecordCount is 5
KADaoDatabase1.StartTransaction;
KADaoTable1.TableType=dbOpenTable;
KADaoTable1.Open;
KADaoTable1.First;
For X := 1 to 1000 do
   Begin
         KADaoTable1.Insert;
         KADaoTable1.Post;
   End;
KADaoTable1.Close;
KADaoDatabase1.Rollback;
//******************* Here KADaoTable1.RecordCount is AGAIN 5
```

KADao fixes this problem but this bug can slowdown your apps at 1000%

To avoid this slowdown you must do the following:

Use dbOpenDynaset (This slows operation by 2-5% but RecordCount always return correct number of records)

```
KADaoTable1.TableType=dbOpenDynaset;
```

or REGULLARY **COMPACT AND REPAIR** your databases.

ALSO ALWAYS COMPACT AND REPAIR DATABASES YOU GET FROM OTHER SOURCES BEFORE USE!

Property SaveUsername:Boolean;

True by default;
When set to True login dialog shows the UserName otherwise UserName is blank
Example:

```
KADaoDatabase1.Database      := 'Demo.mdb';
KADaoDatabase1.UserName      := 'Admin';
KADaoDatabase1.Password      := 'SecretWord';
KADaoDatabase1.LoginPrompt   := True;
KADaoDatabase1.SaveUsername  := False;
KADaoDatabase1.Connected     := True;
{do some stuff ...}
KADaoDatabase1.Connected:=False;
```

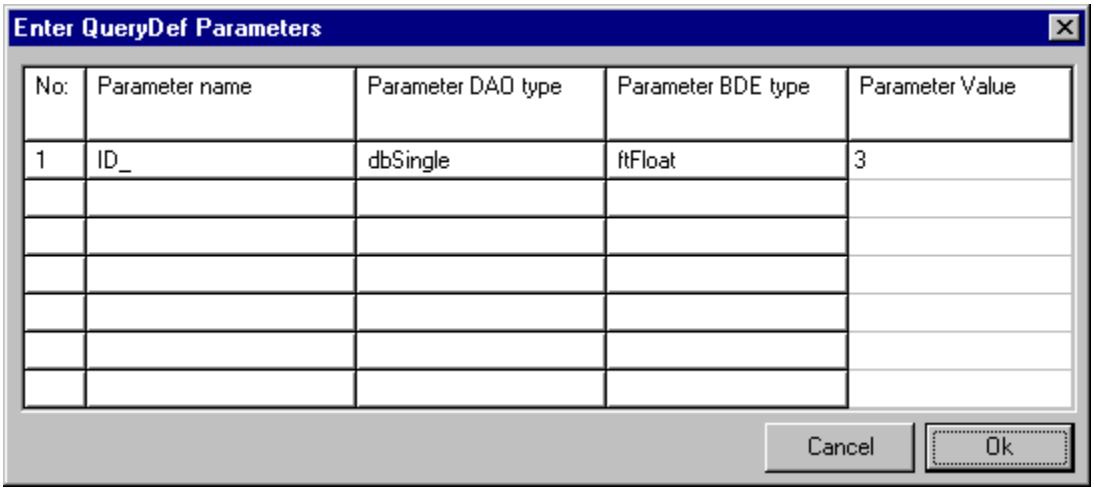
Property UseRecordCount:Boolean;

True by default;
When set to True enables cashing of recordcount which speedsup your application especially with tables different then standard tables (dbOpenTable)
Disable if you want to get always RecordCount from DAO
Example:

```
KADaoDatabase1.Database      := 'Demo.mdb';
KADaoDatabase1.UserName      := 'Admin';
KADaoDatabase1.Password      := 'SecretWord';
KADaoDatabase1.Connected     := True;
KADaoTable1.UseRecordCount   := False;
KADaoTable1.Active           := True;
{do some stuff ...}
KADaoTable1.Active           := False;
KADaoDatabase1.Connected:=False;
```

Function PromptQueryDefParameters:Boolean;

When called brings up the same dialog as QueryDefParameters Editor in design time:



Example:

```
KADaoDatabase1.Database       := 'Demo.mdb';
KADaoDatabase1.UserName       := 'Admin';
KADaoDatabase1.Password       := 'SecretWord';
KADaoDatabase1.Connected      := True;
KADaoTable1.QueryDefName      := 'Query1';
if KADaoTable1.PromptQueryDefParameters Then
     Begin
       KADaoTable1.Active := True;
{do some stuff ...}
KADaoTable1.Active := False;
     End;
KADaoDatabase1.Connected:=False;
```

Stay informed on Dao

Always visit **TechInfo** section of **Delphi Warrior site** for latest news about Microsoft DAO and useful documents and hints

Known bugs and problems

The following methods cannot be used with Filtered Tables based on **OnFilterRecord** event and Tables with **applyed ranges**:

Locate
FindXXX and Find_XXX
Seek_XXX
GotoKey

Another known problem is **using KADao with DataModules**
KADao can be used in DataModules but at application startup all DataAware controls must be relinked to the DataSources.
Still cannot find why this happens.
Example:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  DBGrid1.DataSource:=DataModule2.DataSource1;
  DBGrid2.DataSource:=DataModule2.DataSource2;
end;
```

Also in the dpr file first must be created DataModules containing KADao Databases and Tables and after that Forms containing DBAware controls that use KADao
Example:

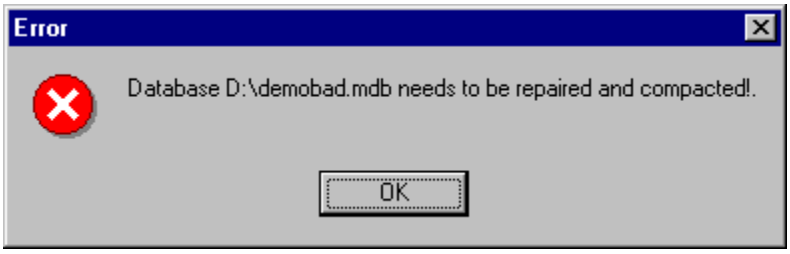
```
begin
  Application.Initialize;
  Application.CreateForm(TDataModule2, DataModule2);
  Application.CreateForm(TDataModule1, DataModule1);
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Note: ***At design time Forms containing DBAware controls must be loaded after DataModules containing KADao components***

Property WarnOnBadDatabase:Boolean;

False by default

When this property is True and KADaoTable founds a table with bad RecordCount an Exception is raised to inform the user that Database needs to be REPAIRED and COMPACTED.



If WarnOnBadDatabase is False KADaoTable handles the error but this will slowdown your application performance.

Property MasterAutoActivate:Boolean;

True by default.
When this property is True if a Detail dataset is set to Active state and the corresponding Master dataset is not active then Detail dataset activates the Master.
Example:

```
 KADaoDatabase1.Database:='master.mdb';
KADaoDatabase1.Connected:=True;
//****************** Setup Master Table properties
KADaoTable1.Database:=KADaoDatabase1;
KADaoTable1.TableName:='CUSTOMER';
Datasource1.DataSet:=KADaoTable1;
DBGrid1.DataSource:=Datasource1;
//****************** Setup Detail Table properties
KADaoTable2.Database:=KADaoDatabase1;
KADaoTable2.TableName:='ORDERS';
KADaoTable2.TableType:=dbOpenDynaset;
KADaoTable2.MasterSource:=Datasource1;
KADaoTable2.MasterFields.Clear;
KADaoTable2.MasterFields.Add('CustNo -> CustNo');
KADaoTable2.MasterAutoActivate:=True;
KADaoTable2.Active:=True;
Datasource2.DataSet:=KADaoTable2;
DBGrid2.DataSource:=Datasource2;
```

How to copy DBGrid selection to Clipboard?

Copying to Clibpoard is very easy with KADao.
The following example copy selected rows (except BLOBS) in a DBGrid to Clipboard as tab delimited text:

Example:

```
procedure TForm1.CopySelectedToCliboard(DBGrid:TDBGrid);
Var
  X,Y       : Integer;
  Clp       : TClipboard;
  S         : String;
  MainTable : TKADaoTable;
begin
 Clp        := TClipboard.Create;
 MainTable  := TKADaoTable(DBGrid.DataSource.DataSet);
 S:='';
 MainTable.DisableControls;
 Try
 For X := 0 To DBGrid.SelectedRows.Count-1 do
     Begin
       MainTable.Bookmark:=DBGrid.SelectedRows.Items[X];
       For Y := 0 To MainTable.FieldDefs.Count-1 do
           Begin
             if NOT MainTable.Fields[Y].ISBlob Then
                Begin
                  S:=S+MainTable.Fields[Y].AsString;
                  if Y < MainTable.FieldDefs.Count-1 Then S :=S+#9;
                    End;
           End;
       S:=S+#13+#10;
     End;
 Finally
   MainTable.EnableControls;
 End;
 If S <> '' Then Clp.AsText:=S;
 Clp.Free;
end;
```

Procedure GoOffline;

Temporary suspends creating of CoreWorkspace so programmer can enter the tree impontant parameters

- Username
- Password
- SystemDatabase

After that you must call GoOnline and RecreateCore to create User's Workspace.

Example 1:

```
      Var
          KADaoDatabase1 : TKADaoDatabase;
      begin
          KADaoDatabase1 := TKADaoDatabase.Create(Self);
          KADaoDatabase1.GoOffline;
          KADaoDatabase1.UserName:='Admin';
          KADaoDatabase1.Password:='SecretWord';
          KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
          KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
          KADaoDatabase1.DatabaseType:='Access';
          KADaoDatabase1.GoOnline;
          KADaoDatabase1.RecreateCore;
          KADaoDatabase1.CreateAccessDatabase('d:\MyDir\runtimex.mdb');
          Message('OK You create an database!');
          KADaoDatabase1.Free;
  end;
```

Procedure GoOnline

Restores standart state of KADaoDatabase after a call to GoOnline.
After that you must call RecreateCore to create User's Workspace.

Example 1:

```
      Var
          KADaoDatabase1 : TKADaoDatabase;
      begin
          KADaoDatabase1 := TKADaoDatabase.Create(Self);
          KADaoDatabase1.GoOffline;
          KADaoDatabase1.UserName:='Admin';
          KADaoDatabase1.Password:='SecretWord';
          KADaoDatabase1.SystemDatabase:='d:\permissions.mwd';
          KADaoDatabase1.DatabasePassword:='AnotherSecretWord';
          KADaoDatabase1.DatabaseType:='Access';
          KADaoDatabase1.GoOnline;
          KADaoDatabase1.RecreateCore;
          KADaoDatabase1.CreateAccessDatabase('d:\MyDir\runtimex.mdb');
          Message('OK You create an database!');
          KADaoDatabase1.Free;
  end;
```

Procedure SetKey;

Sets the find information for GotoKey and GotoNearest method

Example:

```
KADaoTable1.SetKey;
KADaoTable1.FieldByName('AUD').AsString := '1062.6';
KADaoTable1.EditKey;
KADaoTable1.FieldByName('AUD').AsString := '1073.16';
KADaoTable1.GotoKey;
```

Procedure GotoNearest: Boolean;

Same as TTable.GotoNearest
You must call SetKey before using GotoNearest
Returns True if specifyed record is found, false othervise
Example:

```
KADaoTable1.SetKey;
KADaoTable1.FieldByName('AUD').AsString := '1062.6';
KADaoTable1.EditKey;
KADaoTable1.FieldByName('AUD').AsString := '1073.16';
KADaoTable1.GotoNearest;
```

Procedure EditKey;

Changes the find information for GotoKey and GotoNearest method
Find information must be set with SetKey method
Example:

```
KADaoTable1.SetKey;
KADaoTable1.FieldByName('AUD').AsString := '1062.6';
KADaoTable1.EditKey;
KADaoTable1.FieldByName('AUD').AsString := '1073.16';
KADaoTable1.GotoKey;
```

Procedure CancelKey;

Cancels setting find information generateb by SetKey or EditKey and puts Table again in dsBrowse state
Example:

```
KADaoTable1.SetKey;
KADaoTable1.FieldByName('AUD').AsString := '1062.6';
KADaoTable1.EditKey;
KADaoTable1.FieldByName('AUD').AsString := '1073.16';
KADaoTable1.CancelKey;
```

Procedure SaveToStream(Stream:TStream);

Saves Table contents to stream.
The Stream and File formats are compatible with kbmMemTable
created by Kim Bo Madsen - Scandinavia - kbm@optical.dk which is the best MemoryTable i have seen.
Only Data fields are stored. Blobs are stored too.
Use LoadFromBinaryFile and LoadFromBinaryStream methodts of kbmMemTable to Load Datasets saved from KADaoTable.
Using this two methods you can move your data to other Database platforms away from your office.
Example:

```
MS:=TMemoryStream.Create;
KADaoTable1.SaveToStream(MS);
```

Procedure SaveToFile(const FileName:String);

Saves Table contents to file.
The Stream and File formats are compatible with kbmMemTable
created by Kim Bo Madsen - Scandinavia - kbm@optical.dk which is the best MemoryTable i have seen.
Only Data fields are stored. Blobs are stored too.
Use LoadFromBinaryFile and LoadFromBinaryStream methodts of kbmMemTable to Load Datasets saved from KADaoTable.
Using this two methods you can move your data to other Database platforms away from your office.
Example:

```
KADaoTable1.SaveToFile('D:\Table1.bin');
```

Procedure LoadFromStream(Stream:TStream; Mode : TLoadMode);

Load Table contents from stream.

The Stream and File formats are compatible with kbmMemTable created by Kim Bo Madsen - Scandinavia - kbm@optical.dk

which is the best MemoryTable i have seen.

Only Data fields are stored. Blobs are stored too.

Use LoadFromBinaryFile and LoadFromBinaryStream methodts of kbmMemTable to Load Datasets saved from KADaoTable.

Using this two methods you can move your data to other Database platforms away from your office.

Mode describes the way of adding data to table.

TLoadMode = (lmAppend, lmEmptyAppend);

lmAppend just appends the data to the Table.

lmEmptyAppend first reamoves all data to the table and then appends data from the file

Example:

```
KADaoTable1.LoadFromStream(MS, lmAppend);
```

Procedure LoadFromFile(const FileName: String; Mode : TLoadMode);

Load Table contents from file.

The Stream and File formats are compatible with kbmMemTable created by Kim Bo Madsen - Scandinavia - kbm@optical.dk

which is the best MemoryTable i have seen.

Only Data fields are stored. Blobs are stored too.

Use LoadFromBinaryFile and LoadFromBinaryStream methodts of kbmMemTable to Load Datasets saved from KADaoTable.

Using this two methods you can move your data to other Database platforms away from your office.

Mode describes the way of adding data to table.

TLoadMode = (lmAppend, lmEmptyAppend);

lmAppend just appends the data to the Table.

lmEmptyAppend first reamoves all data to the table and then appends data from the file

Example:

```
KADaoTable1.LoadFromFile('D:\Table1.bin', lmAppend);
```

Property OnExportProgress : TExportProgressEvent;
TExportProgressEvent = procedure(Current,Total:Integer) of object;

This event is triggered each time when a new records is SAVED to File or Stream.
Current is zero based position of the current saved record
Total is nuber of records in the table
This event is usefull to implement Progress indicators when exporting large Table to File or Stream
Example:

```
procedure TForm1.KADaoTable1ExportProgress(Current, Total: Integer);
begin
        ProgressBar1.Min:=0;
        ProgressBar1.Max:=Total;
        ProgressBar1.Position:=Current;
end;
```

Property ProcessMessages : Boolean;

False by default.

This property is used to control processing of windows messages when Saving And Loading data to/from File/Stream.

If ProcessMessages is False then all other programs may opearate slowly and windows will not be repaintend until the entire table is Exported/Imported.

Example:

```
KADaoTable1.ProcessMessages:=True;
```

```
//*****************************************************************************************************************
**
        procedure TForm1.KADaoTable1ExportProgress(Current, Total: Integer);
         begin
          KADaoTable1.ProcessMessages:=True; //**** Ensure that ProgressBar will be
        repainted
                 ProgressBar1.Min:=0;
                 ProgressBar1.Max:=Total;
                 ProgressBar1.Position:=Current;
         end;
```

Property UseGetRecNo : Boolean;

True by default.

This property is used to control speed of Filtered datasets or datasets with applyed ranges

When set to False speed is increased but DbGrids does not move vertical scroll bar to show current position in the table.

Also when set to False assigning values or reading from RecNo property does not work.

Example:

```
KADaoTable1.UseGetRecNo:=False;
KADaoTable1.SetRange([3,nil,nil,nil,'3'],[5,nil,nil,nil,'5']);
KADaoTable1.UseGetRecNo:=True;
KADaoTable1.CancelRange;
```

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

Same as TTable method.

Sets the start and end of a range and applyes them to the Table.

Example:

```
KADaoTable1.SetRange([3,nil,nil,nil,'3'],[5,nil,nil,nil,'5']);
```

Procedure SetRangeEnd;

Same as TTable method.
Sets the end of a range.
Example:

```
KADaoTable1.SetRange([3,nil,nil,nil,'3'],[5,nil,nil,nil,'5']);
ShowMessage('Now Range is Active!');
//*************************************************************************
KADaoTable1.SetRangeStart;
KADaoTable1.FieldByName('ID').AsInteger:=3;
KADaoTable1.FieldByName('Tekst').AsString:='3';

KADaoTable1.SetRangeEnd;
KADaoTable1.FieldByName('ID').AsInteger:=5;
KADaoTable1.FieldByName('Tekst').AsString:='5';
KADaoTable1.ApplyRange;
ShowMessage('Now Range is Active!');
//*************************************************************************
KADaoTable1.SetRangeStart;
KADaoTable1.Fields[0].AsInteger:=3;
KADaoTable1.Fields[4].AsString:='3';

KADaoTable1.SetRangeEnd;
KADaoTable1.Fields[0].AsInteger:=5;
KADaoTable1.Fields[4].AsString:='5';
KADaoTable1.ApplyRange;
//*************************************************************************
ShowMessage('Now Range is Active!');
KADaoTable1.CancelRange;
```

Procedure SetRangeStart;

Same as TTable method.
Sets the start of a range.
Example:

```
KADaoTable1.SetRange([3,nil,nil,nil,'3'],[5,nil,nil,nil,'5']);
ShowMessage('Now Range is Active!');
//*************************************************************************
KADaoTable1.SetRangeStart;
KADaoTable1.FieldByName('ID').AsInteger:=3;
KADaoTable1.FieldByName('Tekst').AsString:='3';

KADaoTable1.SetRangeEnd;
KADaoTable1.FieldByName('ID').AsInteger:=5;
KADaoTable1.FieldByName('Tekst').AsString:='5';
KADaoTable1.ApplyRange;
ShowMessage('Now Range is Active!');
//*************************************************************************
KADaoTable1.SetRangeStart;
KADaoTable1.Fields[0].AsInteger:=3;
KADaoTable1.Fields[4].AsString:='3';

KADaoTable1.SetRangeEnd;
KADaoTable1.Fields[0].AsInteger:=5;
KADaoTable1.Fields[4].AsString:='5';
KADaoTable1.ApplyRange;
//*************************************************************************
ShowMessage('Now Range is Active!');
KADaoTable1.CancelRange;
```

Procedure EditRangeEnd;

Same as TTable method.
Edits the end of a range.
Example:

```
//*****************************************************************************
KADaoTable1.SetRangeStart;
KADaoTable1.FieldByName('ID').AsInteger:=3;
KADaoTable1.FieldByName('Tekst').AsString:='3';
KADaoTable1.SetRangeEnd;
KADaoTable1.FieldByName('ID').AsInteger:=5;
KADaoTable1.FieldByName('Tekst').AsString:='5';
KADaoTable1.ApplyRange;
ShowMessage('Now Rage is Active!');
//*****************************************************************************
KADaoTable1.EditRangeStart;
KADaoTable1.Fields[0].AsInteger:=1;
KADaoTable1.Fields[4].AsString:='1';
KADaoTable1.EditRangeEnd;
KADaoTable1.Fields[0].AsInteger:=4;
KADaoTable1.Fields[4].AsString:='4';
KADaoTable1.ApplyRange;
//*****************************************************************************
ShowMessage('Now Rage is Active!');
KADaoTable1.CancelRange;
ShowMessage('Now Rage is Inactive!');
```

Procedure ApplyRange;

Same as TTable method.
Applyes a range;
Example:

```
KADaoTable1.SetRange([3,nil,nil,nil,'3'],[5,nil,nil,nil,'5']);
ShowMessage('Now Range is Active!');
//*************************************************************************
KADaoTable1.SetRangeStart;
KADaoTable1.FieldByName('ID').AsInteger:=3;
KADaoTable1.FieldByName('Tekst').AsString:='3';

KADaoTable1.SetRangeEnd;
KADaoTable1.FieldByName('ID').AsInteger:=5;
KADaoTable1.FieldByName('Tekst').AsString:='5';
KADaoTable1.ApplyRange;
ShowMessage('Now Range is Active!');
//*************************************************************************
KADaoTable1.SetRangeStart;
KADaoTable1.Fields[0].AsInteger:=3;
KADaoTable1.Fields[4].AsString:='3';

KADaoTable1.SetRangeEnd;
KADaoTable1.Fields[0].AsInteger:=5;
KADaoTable1.Fields[4].AsString:='5';
KADaoTable1.ApplyRange;
//*************************************************************************
ShowMessage('Now Range is Active!');
KADaoTable1.CancelRange;
ShowMessage('Now Rage is Inactive!');
```

Procedure CancelRange;

Same as TTable method.
Cancels a range.
Example:

```
KADaoTable1.SetRange([3,nil,nil,nil,'3'],[5,nil,nil,nil,'5']);
ShowMessage('Now Range is Active!');
KADaoTable1.CancelRange;
ShowMessage('Now Rage is Inactive!');
```

Procedure EditRangeStart;

Same as TTable method.
Edits the start of a range.
Example:

```
//*****************************************************************************
KADaoTable1.SetRangeStart;
KADaoTable1.FieldByName('ID').AsInteger:=3;
KADaoTable1.FieldByName('Tekst').AsString:='3';
KADaoTable1.SetRangeEnd;
KADaoTable1.FieldByName('ID').AsInteger:=5;
KADaoTable1.FieldByName('Tekst').AsString:='5';
KADaoTable1.ApplyRange;
ShowMessage('Now Rage is Active!');
//*****************************************************************************
KADaoTable1.EditRangeStart;
KADaoTable1.Fields[0].AsInteger:=1;
KADaoTable1.Fields[4].AsString:='1';
KADaoTable1.EditRangeEnd;
KADaoTable1.Fields[0].AsInteger:=4;
KADaoTable1.Fields[4].AsString:='4';
KADaoTable1.ApplyRange;
//*****************************************************************************
ShowMessage('Now Rage is Active!');
KADaoTable1.CancelRange;
ShowMessage('Now Rage is Inactive!');
```

Property UseCaptions:Boolean;

False by Default
Quering some field properties is extremely slow with MS Dao
This property controls DisplayLabels of Fileds which is equal to MS Access Caption property
When set to True DisplayLabels are retrieved from the Caption property orhervise DisplayLabels are set to Field names
Example:

```
//*************************************************************************
KADaoTable1.UseCaptions=True;
KADaoTable1.Open;
ShowMessage('Now Captions are visible!');
KADaoTable1.Close;
//*************************************************************************
KADaoTable1.UseCaptions=False;
KADaoTable1.Open;
ShowMessage('Now Captions are NOT visible!');
KADaoTable1.Close;
//*************************************************************************
```

Property UseDaoProperties:Boolean;

True by Default
Quering some field properties is extremely slow with MS Dao
This property controls some Fileds properties which can make easy adding new records
When set to False, Default Values are not shown when adding new records
Also Required property is not set on the fields that are required
Also you can modify fields that cannot be modified (this will raise exception on Post)
Setting this property to False will increase speed of opening Queries about 10000% but You must do coding carrefully!
Example:

```
//*************************************************************************
KADaoTable1.UseDaoProperties=True;
KADaoTable1.Open;
ShowMessage('Now DaoProperties are retrieved!');
KADaoTable1.Close;
//*************************************************************************
KADaoTable1.UseDaoProperties=False;
KADaoTable1.Open;
ShowMessage('Now DaoProperties are NOT retrieved!');
KADaoTable1.Close;
//*************************************************************************
```

Property BatchMode:Boolean;

False by Default

Using this property increases the speed of Insering or appending records at about 500%

Use only with appending/inserting multiple records to the table.

BatchMode temporary disables DB contols so you must put `BatchMode:=False` in Try/Finally statement

Example:

```
//*****************************************************************************
KADaoTable1.First;
Try
 KADaoTable1.BatchMode:=True;
 For X := 0 to 1000 do
          Begin
           KADaoTable1.Append;
           KADaoTable1['ID']:=X;
           KADaoTable1['Date']:=Now;
           KADaoTable1['Tekst']:=IntToStr(X);
           KADaoTable1['Small']:=X;
           KADaoTable1['Bait']:=Byte(X);
           KADaoTable1['Test']:=Boolean(Random(2));
           KADaoTable1['Memo']:=IntToStr(X);
           KADaoTable1.Post;
          End;
Finally
     KADaoTable1.BatchMode:=False;
End;
//*****************************************************************************
```

Property CacheMemos:Boolean;

False by Default

Using this property reduces the speed of your application.

Since there are too many dbGrids showing contents of memo fileds (***which violates Borland standarts***) i put this property
to handle such situations.

Set to True ONLY if you will display Memos in dbGrids othervise set to False - this will speedup KADAO.

Example:

```
//*************************************************************************
KADaoTable1.CacheMemos:=True;
//*************************************************************************
```

Property OnImportProgress : TImportProgressEvent;
TImportProgressEvent = procedure(Current:Integer) of object;

This event is triggered each time when a new records is Loaded from File or Stream.
Current is zero based position of the current Loaded record
This event is usefull to implement Progress indicators when importing large Table from File or Stream
Example:

```
procedure TForm1.KADaoTable1ExportProgress(Current, Total: Integer);
begin
        ProgressBar1.Min:=0;
        ProgressBar1.Position:=Current;
end;
```

Property MdbVersionAutoDetect:Boolean;

False by Default

When this property is True and DYNADAO is used then KADao automatically selects DAO 3.6 (Version property='3.6') if

MDB is Access 4.0 file. If MDB is Access 3.x file KADao lives Version property withouth change.

Example:

```
//*****************************************************************************
KADaoTable1.MdbVersionAutoDetect:=False;
//*****************************************************************************
```

Property CacheBlobs:Boolean;

False by Default

Using this property reduces the speed of your application.

Since there are too many dbGrids showing contents of blob images(***which violates Borland standarts***) i put this property
to handle such situations.

Set to True ONLY if you will display Images/Blobs in dbGrids othervise set to False - this will speedup KADAO.

Example:

```
//*************************************************************************
KADaoTable1.CacheBlobs:=True;
//*************************************************************************
```

What contain files in the package?

This is a list of all files included in KADao Package adw what each of them include.

Readme Files - read them before installation

```
-------------------------------------------------------------------------------
Readme.txt              KADao Components Readme file
KADaoInfo Readme.txt    KADaoInfo Component Readme file
```

Support Files

```
-------------------------------------------------------------------------------
DaoApi.pas              Common definitions for both DAO 3.5 and DAO 3.6
                        (Including them in uses clause is a good practice on
DYNADAO)
DAO35Api.pas            Definitions for DAO 3.50
DAO351Api.pas           Definitions for DAO 3.51
DAO36Api.Pas            Definitions for DAO 3.60
DaoUtils.pas            Some conversion routines from DAO to BDE and vice
versa
ErrLangDB.pas           Error messages for KADaoDatabase
ErrLangTB.pas           Error messages for KADaoTable
```

Package Files

```
-------------------------------------------------------------------------------
KADao.bpk               Package file for CBuilder 3/4
KADao.cpp               C definitions for CBuilder 3/4
KADao.res               Resource file for CBuilder 3/4
KADao.dpk               Package file for Delphi 3,4,5
KADaoC5.bpk             Package file for CBuilder 5
KADaoC5.cpp             C definitions for CBuilder 5
KADaoC5.res             Resource file for CBuilder 5
```

Help Files

```
-------------------------------------------------------------------------------
KADao.chm               Help in HTML help format
KADao.cnt               Help Contents in WinHelp format
KADao.hlp               Help in WinHelp format
```

Main Files

```
-------------------------------------------------------------------------------
CommonDirectives.pas    Preprocessor definitions for both KADaoDatabase and
KADaoTable
KDaoDatabase.dcr        Image resources for IDE
KDaoDataBase.pas        KADaoDatabase Component
KDaoTable.pas           KADaoTable Component
KADaoInfo.pas           KADaoInfo Component
KADaoDummyDataset.pas   Support file for KADaoTableManager
```

Support Dialog boxes

```
-------------------------------------------------------------------------------
DBLoginUnit.dfm               Login Dialog form
DBLoginUnit.pas               Login Dialog unit
MasterDetailFormUnit.dfm      Master/Detail Dialog form
MasterDetailFormUnit.pas      Master/Detail Dialog unit
ODBCDialogUnit.dfm            ODBC SOurce selcetion form
ODBCDialogUnit.pas            ODBC SOurce selcetion unit
```

QueryDefDialogUnit.dfm	QueryDef Parameters Dialog form
0QueryDefDialogUnit.pas	QueryDef Parameters Dialog unit
SortByDialog.dfm	SortOrder Dialog form
SortByDialog.pas	SortOrder Dialog unit

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

Same as TTable.FindKey
Uses DAO seek method for fast positioning at record containing values stored in KeyValues array.
Internaly FindKeyEx calls Seek_NearestEx(KeyValues,'>=');
Example:

```
KADaoTable1.IndexName='MyIndex';
KADaoTable1.FindKeyEx([Edit2.Text]);
```

Procedure Sort;

Executes a sorting after SortedBy property is set
You **cannot** use it with StandardTable type;
The following example opens table Table1 sorted first by Field **ID** in ascending order and then by Field **Name** in descending order and executes the Sort method.

Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.TableType      := dbOpenDynaset;
KADaoTable1.Open;
KADaoTable1.SortedBy.Clear;
KADaoTable1.SortedBy.Add('[ID] ASC');
KADaoTable1.SortedBy.Add('[Name] DESC');
KADaoTable1.Sort;
{do some stuff ...}
KADaoTable1.Close;
```

How to ensure that a table is open?

There is no good way to determine whether DAO is busy or not. If you attempt to open a (new) table when DAO is busy, you will get an exception.
A very good example is given by Mark Hamilton:

```
Procedure OpenTable(TableName : string; Database : TKaDaoDatabase; Table :
KaDaoTable);
Fnction CanOpen(TableName : string; Database : TKaDaoDatabase; Table :TKaDaoTable) :
Boolean;
begin
    Result := True;
    try
      Table.Database := Database;
      Application.ProcessMessages;
      Database.Idle;
      try
        Table.TableName := TableName;
        Application.ProcessMessages;
        Database.Idle;
        try
          Table.Open;
          Application.ProcessMessages;
          Database.Idle;
        except
          Result := False;
        end;
      except
        Result := False;
      end;
    except
      Result := False;
    end;
end;
begin // Procedure
    if (Table.Active) then
        Table.Close;
    repeat
         Application.ProcessMessages;
         Database.Idle;
         Sleep(0);
    until (CanOpen(TableName, Database, Table));
end;
```

Note:
The number and placement of the Idle commands is the key to ensuring that exceptions are trapped in a timely fashion and allow
for latency induced by DAO if it is busy when the method call is made.

Default Values Handling

Default values of fields can be hadled in two ways.

First you can specify default value for a field directly in the database using a tool as MSAccess or KADaoField component from KADaoDeluxe package.
Second you can specify default value using DefaultExpression property of each field of KADaoTable. In this case DefaultExpression has bigger priority then the DefaultValue property of the MS Access Field.

Mark Hamilton's Column about enchancing your work with DAO and KADao

Mark Hamilton (MarkHamilton@compuserve.com) has a very deep investigations on enchancing your work with boot MS DAO and KADao. You must know about some very interesting things he foud. I am publishing them (with his permission) without any corrections and with great pleasure.

DAO can be a real memory hog. These tips will help you reduce the amount of memory your app seems to consume:

* Avoid opening Tables of type "dbOpenTable" (i.e. StandardTable) unless it is essential you access records via an index. But it is much faster to add records to a StandardTable than it is to a Dynaset, particularly if the Table's BatchMode property is set to True.

* Indexes consume much memory. Create the indexes but use Dynaset table-types ordered on fields that are indexed. Internally, "Rushmore" technology uses indexes where it can in order to optimise sort sequences for non-dbOpenTable tables.

* Provided your tables are reasonably-sized, it is much more memory-efficient to open them via a SQL SELECT statement and load the records into a (dynamically-created) kbmMemTable using either KaDaoTable.SaveToStream and kbmMemTable.LoadFromBinaryStream or kbmMemTable's CopyRecords procedure; then close the Dynaset. You can then access your data either in an indexed order or use a sort sequence.

* If you then need to edit records, set kbmMemTable's RecordTag to a non-zero value for those records whose values have changed. Then to make the changes permanent, create a loop to generate SQL UPDATE commands, viz:

```
      MemTable.First;
        SQLCmds := TStringList.Create;
        try
          while (not MemTable.Eof) do
          begin
              if (MemTable.RecordTag <> 0) then
              begin
                  SQLCmds.Clear;
                  SQLCmds.Add(Format('UPDATE [%s] SET [Field1]=%s,[Field2]=
%d',
                                          [MyTableName,
MemTable.FieldByName('StringVariable').AsString,

MemTable.FieldByName('NumericVariable').AsInteger]));
                  SQLCmds.Add(Format('WHERE [KeyField]=%d',

[MemTable.FieldByName('NumericKey').AsInteger]));
                  DynaTable.ExecSQL(SQLCmds);
                  Application.ProcessMessages;
                  KaDaoDatabase1.Idle;
              end;
              MemTable.Next;
          end;
        finally
          SQLCmds.Free;
        end;
```

Not only will this execute much faster than if you open the MDB file's table, seek to each record, edit and then post, but you will save (potentially megabytes of) memory too. (Please note that you should enclose Table names and Field names within square brackets for all SQL commands and, to avoid Variant Conversion errors, you should not use the shorthand "MemTable['FieldName']" when assigning field values using the Format function.)

* Most memory is consumed whenever the Table's state changes from dsBrowse to dsEdit or dsAdd. DAO is not very good about recycling record buffers: it prefers to allocate new ones each time.

* If memory management is a real concern, then investigate the excellent freeware class "QMemory" by Andrew Driazgov. If you do use this to replace Delphi's default, and you dynamically-create your KaDaoDatabase and KaDaoTable components, then you can do something along the following lines:

```
        try
          KaDaoDatabase.CloseDataSets;    // Causes an exception if no tables
are open
        except
        end;

        KaDaoDatabase.Close;
        KaDaoTable.Free;
        KaDaoDatabase.Free;

        for i := 0 to 5 do
        begin
            QMemory.QMemDecommitOverstock;
            Application.ProcessMessages;
            Sleep(0);
        end;

        KaDaoDataBase := TKaDaoDatabase.Create(Self);
        ....                              // Set database properties
        KaDaoDatabase.Open;
        KaDaoTable := TKaDaoTable.Create(Self);
        KaDaoTable.Database := KaDaoDataBase;
        ....                              // Set remaining table properties
        KaDaoTable.FieldDefs.Clear;
        KaDaoTable.FieldDefs.Add(.....)  // Create the TField descendent
objects
        ....
        MyDataSource.DataSet := KaDaoTable;
```

This will free around 90-95% of all the memory DAO uses without unloading the whole DAO sub-system or disturbing the COM link. Unfortunately, you can not achieve this without using QMemory as it organises memory in a different way to the standard memory manager. Under Windows '95/'98, the Database Engine must be **Private** for this scheme to work. Under NT/2000, the Database Engine can be either Private or Public but better results will be achieved if it is **Public**.

* If your application makes extensive use of textual memo fields and these are likely to be 4kb or greater, then create dbVersion30 (and NOT dbVersion40) databases to store your data. dbVersion30 (Access '97) stores textual memo fields as one byte per character but dbVersion40 (Access '2000) stores those fields

in Unicode format - two bytes per character. Access '2000 does define a special property for these fields: "Unicode Compression" but this will only work if it will result in a memo of less than 4096 bytes - and even this is not guaranteed. (Note this is not a TMemoField property so can not be modified programmatically.) Unicode'd memos are particular memory hogs since two buffers must be allocated for them (four if they're edited). They also cause your MDB file to occupy considerably more disk space than is necessary. (Sidebar: if you have created a dbVersion40 database and, as a result of reading this note, you want to convert it to a dbVersion30 one, then please bear in mind that DAO's RepairDatabase methods will NOT un-UniCode memo fields. Having converted the database, use KaDaoTable's SaveToFile, delete and recreate the table, finally re-populate it using KaDaoTable's LoadFromFile.)

Procedure RepairAccessDatabaseEx(DatabaseName : String;
NewLocale : String;
Encrypt : Boolean;
Decrypt : Boolean;
NewVersion : Integer;
Password : String);

This is a more detailed and preffered method to repair a Microsoft Access database.
It gives an opportunity to change some database settings.
Also it can be used to change the format of the database to an old version format.
Note that this function can be used only on DAO 3.6 Engine. On Dao3.5 this procedure simply calls RepairAccessDatabase.
DatabaseName is the full pathname of the mdb file to be compacted;
NewLocale is one of the following constants predefined in DaoApi.pas:

```
dbLangArabic;
dbLangCzech;
dbLangDutch;
dbLangGeneral;
dbLangGreek;
dbLangHebrew;
dbLangHungarian;
dbLangIcelandic';
dbLangNordic;
dbLangNorwDan;
dbLangPolish;
dbLangCyrillic;
dbLangSpanish;
dbLangSwedFin;
dbLangTurkish;
dbLangJapanese;
dbLangChineseSimplified;
dbLangChineseTraditional;
dbLangKorean;
dbLangThai;
```

Encrypt is a boolean variable. If set to true the compacted database will be encrypted.
Decrypt is a boolean variable. If set to true the compacted database will be decrypted.
If both Encrypt and Decrypt are true Database will be Decrypted
NewVersion is the new version of the compacted database.
There are a few predefined versions in DaoApi.pas:

```
dbVersion11;
dbVersion20;
dbVersion30;
dbVersion40;
```

Remember that you cannot create version40 database with Dao35

Password is a global database password if database have one.
If Database does NOT have password set Password to empty string
If Database does NOT have password and Password is not empty string the compacted database will have global password same as Password parameter.

Example:

```
KADaoDatabase1.RepairAccessDatabaseEx('d:\www\runtimex.mdb','',False,True,0,'');
```

Function CreateQueryDef(Name:String;SQL:String):Boolean;

Modifyes an existing QueryDef object stored in the database.
Name is the unique name of the QueryDef object
SQL contains the Query definition.
Example:

```
KADaoDatabase1.Database:='Demo.mdb';
KADaoDatabase1.Open;
KADaoDatabase1.CreateQueryDef('NewQueryDef','Select * from [MyTable];';
KADaoDatabase1.ModifyQueryDef('NewQueryDef','Select ID from [MyTable];';
KADaoDatabase1.Close;
```

Procedure AccessExportToTXT(FileName:String; IncludeBlobs, DeleteOld:Boolean);

This method exports the table to the text file.

Table must be opened before you can use this method!
FileName is the full path to the file to which table will be exported;
InclubeBlobs when True causes method to export BLOB fields too otherwise BLOB fields are not exported;
DeleteOld when True deletes an existing file with the same name if a such file exists. Otherwise if file exists and DeleteOld is False an eception is raised.

When exporting to a text file it is a goot practice to cerete a **schema.ini** file in the same directory where text file will be exported
Contents of a simple **schema.ini** follows:

```
[Table1.txt]
ColNameHeader=True
CharacterSet=ANSI
Format=Delimited(;)

    [Table2.txt]
ColNameHeader=False
CharacterSet=OEM
Format=Delimited(;)
```

Limitations:
All Table rows is exported no matter what filters based on OnFilter event are applyed.
However filters based on Filter property are applyed to the result table
Tables based on parametrized QueryDefs are not exported.
Only those fields that are incliuded with fields editor are exported.

Example:

```
Var
   KADaoTable1     : TKADaoTable;
 Begin
   KADaoTable1            := TKADaoTable.Create(Self);
   {set properties such Database etc ...}
   {...}
   KADaoTable1.TableName  := 'Table1';
   KADaoTable1.Open;
   KADaoTable1.AccessExportToTXT('d:\tests\Table1.txt',True, True);
   KADaoTable1.AccessExportToHTML('d:\tests\Table1.htm',True, True);
   KADaoTable1.AccessExportToExcel('d:\tests\Table1.xls','Tab1',3,True, True);
   KADaoTable1.AccessExportToParadox('d:\tests\Table1.db',7,True, True);
   KADaoTable1.AccessExportToDBase('d:\tests\Table1.dbf',5,True, True);
   KADaoTable1.AccessExportToMDB('d:\tests\demo.mdb','NewTable',True, True);
   KADaoTable1.Close;
   KADaoTable1.Free;
 End;
```

Procedure AccessExportToHTML(FileName:String; IncludeBlobs, DeleteOld:Boolean);

This method exports the table to the HTML file.

Table must be opened before you can use this method!
FileName is the full path to the file to which table will be exported;
InclubeBlobs when True causes method to export BLOB fields too otherwise BLOB fields are not exported;
DeleteOld when True deletes an existing file with the same name if a such file exists. Otherwise if file exists and DeleteOld is False an eception is raised.

When exporting to a HTML file it is a goot practice to cerete a **schema.ini** file in the same directory where HTML file will be exported
Contents of a simple **schema.ini** follows:

```
[Table1.htm]
ColNameHeader=True
CharacterSet=ANSI
Format=HTML

    [Table2.txt]
ColNameHeader=False
CharacterSet=OEM
Format=HTML
```

Limitations:
All Table rows is exported no matter what filters based on OnFilter event are applyed.
However filters based on Filter property are applyed to the result table
Tables based on parametrized QueryDefs are not exported.
Only those fields that are incliuded with fields editor are exported.

Example:

```
Var
   KADaoTable1     : TKADaoTable;
 Begin
   KADaoTable1            := TKADaoTable.Create(Self);
   {set properties such Database etc ...}
   {...}
   KADaoTable1.TableName  := 'Table1';
   KADaoTable1.Open;
   KADaoTable1.AccessExportToTXT('d:\tests\Table1.txt',True, True);
   KADaoTable1.AccessExportToHTML('d:\tests\Table1.htm',True, True);
   KADaoTable1.AccessExportToExcel('d:\tests\Table1.xls','Tab1',3,True, True);
   KADaoTable1.AccessExportToParadox('d:\tests\Table1.db',7,True, True);
   KADaoTable1.AccessExportToDBase('d:\tests\Table1.dbf',5,True, True);
   KADaoTable1.AccessExportToMDB('d:\tests\demo.mdb','NewTable',True, True);
   KADaoTable1.Close;
   KADaoTable1.Free;
 End;
```

Procedure AccessExportToExcel(FileName, SheetName :String; ExcelVersion:Integer; IncludeBlobs, DeleteOld:Boolean);

This method exports the table to the MS Excel file.

Table must be opened before you can use this method!
FileName is the full path to the file to which table will be exported.
SheetName is the name of the sheet to be created in the xls file.
InclubeBlobs when True causes method to export BLOB fields too otherwise BLOB fields are not exported;
DeleteOld when True deletes an existing file with the same name if a such file exists. Otherwise if file exists and DeleteOld is False an eception is raised (only if the SheetName already exists).
ExcelVersion is an integer describing the version of xls file to be created.

Limitations:
All Table rows is exported no matter what filters based on OnFilter event are applyed.
However filters based on Filter property are applyed to the result table
Tables based on parametrized QueryDefs are not exported.
Only those fields that are incliuded with fields editor are exported.

Example:

```
Var
   KADaoTable1     : TKADaoTable;
 Begin
   KADaoTable1            := TKADaoTable.Create(Self);
   {set properties such Database etc ...}
   {...}
   KADaoTable1.TableName  := 'Table1';
   KADaoTable1.Open;
   KADaoTable1.AccessExportToTXT('d:\tests\Table1.txt',True, True);
   KADaoTable1.AccessExportToHTML('d:\tests\Table1.htm',True, True);
   KADaoTable1.AccessExportToExcel('d:\tests\Table1.xls','Tab1',3,True, True);
   KADaoTable1.AccessExportToParadox('d:\tests\Table1.db',7,True, True);
   KADaoTable1.AccessExportToDBase('d:\tests\Table1.dbf',5,True, True);
   KADaoTable1.AccessExportToMDB('d:\tests\demo.mdb','NewTable',True, True);
   KADaoTable1.Close;
   KADaoTable1.Free;
 End;
```

Procedure AccessExportToParadox(FileName :String; ParadoxVersion:Integer; IncludeBlobs, DeleteOld:Boolean);

This method exports the table to the Paradox Table

Table must be opened before you can use this method!
FileName is the full path to the folder to which table will be exported.
InclubeBlobs when True causes method to export BLOB fields too otherwise BLOB fields are not exported;
DeleteOld when True deletes an existing file with the same name if a such file exists. Otherwise if file exists and DeleteOld is False an eception is raised.
ParadoxVersion is an integer describing the version of Paradox Table to be created.

Limitations:
All Table rows is exported no matter what filters based on OnFilter event are applyed.
However filters based on Filter property are applyed to the result table
Tables based on parametrized QueryDefs are not exported.
Only those fields that are incliuded with fields editor are exported.

Example:

```
Var
  KADaoTable1     : TKADaoTable;
Begin
  KADaoTable1            := TKADaoTable.Create(Self);
  {set properties such Database etc ...}
  {...}
  KADaoTable1.TableName  := 'Table1';
  KADaoTable1.Open;
  KADaoTable1.AccessExportToTXT('d:\tests\Table1.txt',True, True);
  KADaoTable1.AccessExportToHTML('d:\tests\Table1.htm',True, True);
  KADaoTable1.AccessExportToExcel('d:\tests\Table1.xls','Tab1',3,True, True);
  KADaoTable1.AccessExportToParadox('d:\tests\Table1.db',7,True, True);
  KADaoTable1.AccessExportToDBase('d:\tests\Table1.dbf',5,True, True);
  KADaoTable1.AccessExportToMDB('d:\tests\demo.mdb','NewTable',True, True);
  KADaoTable1.Close;
  KADaoTable1.Free;
End;
```

Procedure AccessExportToDBase(FileName :String; DBaseVersion:Integer; IncludeBlobs, DeleteOld:Boolean);

This method exports the table to the dBase Table

Table must be opened before you can use this method!
FileName is the full path to the folder to which table will be exported.
InclubeBlobs when True causes method to export BLOB fields too otherwise BLOB fields are not exported;
DeleteOld when True deletes an existing file with the same name if a such file exists. Otherwise if file exists and DeleteOld is False an eception is raised.
DBaseVersion is an integer describing the version of dBase Table to be created.

Limitations:
All Table rows is exported no matter what filters based on OnFilter event are applyed.
However filters based on Filter property are applyed to the result table
Tables based on parametrized QueryDefs are not exported.
Only those fields that are incliuded with fields editor are exported.

Example:

```
Var
  KADaoTable1     : TKADaoTable;
Begin
  KADaoTable1            := TKADaoTable.Create(Self);
  {set properties such Database etc ...}
  {...}
  KADaoTable1.TableName  := 'Table1';
  KADaoTable1.Open;
  KADaoTable1.AccessExportToTXT('d:\tests\Table1.txt',True, True);
  KADaoTable1.AccessExportToHTML('d:\tests\Table1.htm',True, True);
  KADaoTable1.AccessExportToExcel('d:\tests\Table1.xls','Tab1',3,True, True);
  KADaoTable1.AccessExportToParadox('d:\tests\Table1.db',7,True, True);
  KADaoTable1.AccessExportToDBase('d:\tests\Table1.dbf',5,True, True);
  KADaoTable1.AccessExportToMDB('d:\tests\demo.mdb','NewTable',True, True);
  KADaoTable1.Close;
  KADaoTable1.Free;
End;
```

Property Encrypter:TComponent;

This property gives the opportunity for the programmer to use an external component for Encrypting Strings, Memos and Blobs.
A simple Encrypter component comes with KADao and does NOTHING!
You must write code in the two supplyed routines in KADaoEncrypter:

```
Function  TKADAOEncrypter.EncodeString(Value:String):String;
Begin
 Result := Value;
 //********************************** WRITE YOUR CUSTOM ENCRYPTION CODE HERE
********************
End;

Function  TKADAOEncrypter.DecodeString(Value:String):String;
Begin
 Result := Value;
 //********************************** WRITE YOUR CUSTOM DECRYPTION CODE HERE
********************
End;
```

Property DatabaseAutoActivate:Boolean;

False by Default

When this property is True and corresponding KADaoDatabase is not connected KADaoTable first connect the database and the tryes to open the recordset.

When this property is False an exeption informing that corresponding KADaoDatabase is not connected.

Example:

```
//*************************************************************************
KADaoTable1.DatabaseAutoActivate:= True;
KADaoTable1.Database.Connected  : =False;
KADaoTable1.Open;
//*************************************************************************
```

Procedure LinkExternalTable(Database,TableName,TableFileName,TableType:String;TableAttributes:Integer);

Links an external table to a Database;
Database is the name of database and depends on TableType parameter.
If for example you want to link a MicrosoftAccess table from another Database ypu must pass something like 'c:\mydir\mymdb.mdb';
TableName is the Name of the new linked table to be created.
TableFileName is the FileName of the table from external database.
TableType is one of predefined TableTypes in DaoApi.Pas

```
{ _PredefinedTableTypes}
  dBase_50_Table        = 'dBase 5.0;DATABASE=%s';     {Drive:\Path}
  dBase_III_Table       = 'dBase III;DATABASE=%s';     {Drive:\Path}
  dBase_IV_Table        = 'dBase IV;DATABASE=%s';      {Drive:\Path}
  Excel_30_Table        = 'Excel 3.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_40_Table        = 'Excel 4.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_50_Table        = 'Excel 5.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Excel_80_Table        = 'Excel 8.0;DATABASE=%s';     {Drive:\Path\
Filename.xls}
  Exchange_40_Table     = 'Exchange 4.0;MAPILEVEL=$s'; {Brrrrrrrrrr}
  HTML_Import_Table     = 'HTML Import;DATABASE=%s';   {URL}
  Jet_Table             = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Jet_2x_Table          = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Jet_3x_Table          = ';DATABASE=%s';              {Drive:\Path\
Filename.mdb}
  Lotus_WK1_Table       = 'Lotus WK1;DATABASE=%s';     {Drive:\Path\
Filename.wk1}
  Lotus_WK3_Table       = 'Lotus WK3;DATABASE=%s';     {Drive:\Path\
Filename.wk3}
  Lotus_WK4_Table       = 'Lotus WK4;DATABASE=%s';     {Drive:\Path\
Filename.wk4}
  FoxPro_20_Table       = 'FoxPro 2.0;DATABASE=%s';    {Drive:\Path}
  FoxPro_25_Table       = 'FoxPro 2.5;DATABASE=%s';    {Drive:\Path}
  FoxPro_26_Table       = 'FoxPro 2.6;DATABASE=%s';    {Drive:\Path}
  FoxPro_30_Table       = 'FoxPro 3.0;DATABASE=%s';    {Drive:\Path}
  Paradox_3X_Table      = 'Paradox 3.X;DATABASE=%s';   {Drive:\Path}
  Paradox_4X_Table      = 'Paradox 4.X;DATABASE=%s';   {Drive:\Path}
  Paradox_5X_Table      = 'Paradox 5.X;DATABASE=%s';   {Drive:\Path}
  Paradox_7X_Table      = 'Paradox 7.X;DATABASE=%s';   {Drive:\Path}
  Text_Table            = 'Text;DATABASE=%s';          {Drive:\Path}
  ODBC_Table            = 'ODBC;DATABASE=%s;UID=%s;PWD=%s;DSN=%s';
  ODBC_Table_Prompt    =  'ODBC;';
```

TableAttributes is an integer variable which contain various link options used by DAO
It can be a combination of the following constants predefined in DaoApi.pas

```
dbAttachExclusive;
dbAttachSavePWD;
dbSystemObject;
dbAttachedTable;
dbAttachedODBC;
dbHiddenObject;
```

See DAO help for descrptions of these constants.

Example:

```
{Link an Excel table}
  KADaoTable1.Active     := False;
```

```
  KADaoTable1.TableName := '';
  KADaoDatabase1.LinkExternalTableEx('d:\
Temp\','TestMaster','TestMaster.txt','Text;DATABASE=%s',0);
  KADaoTable1.TableName := 'TestMaster';
  KADaoTable1.Active    := True;
```

Procedure AccessExportToMDB(FileName,NewTableName:String; IncludeBlobs, DeleteOld:Boolean);

This method exports the table to external Microsoft Access Database (MDB) file.

Table must be opened before you can use this method!
FileName is the full path to the MDB file to which table will be exported;
NewTableName is the name of the table to which data will be exported.
InclubeBlobs when True causes method to export BLOB fields too otherwise BLOB fields are not exported;
DeleteOld when True deletes an existing table with the same name if a such table exists. Otherwise if table exists and DeleteOld is False an eception is raised.

Limitations:
MDB file must existst.
All Table rows is exported no matter what filters based on OnFilter event are applyed.
However filters based on Filter property are applyed to the result table
Tables based on parametrized QueryDefs are not exported.
Only those fields that are incliuded with fields editor are exported.

Example:

```
Var
   KADaoTable1     : TKADaoTable;
 Begin
   KADaoTable1             := TKADaoTable.Create(Self);
   {set properties such Database etc ...}
   {...}
   KADaoTable1.TableName  := 'Table1';
   KADaoTable1.Open;
   KADaoTable1.AccessExportToMDB('d:\tests\demo.mdb','NewTable',True, True);
   KADaoTable1.AccessExportToTXT('d:\tests\Table1.txt',True, True);
   KADaoTable1.AccessExportToHTML('d:\tests\Table1.htm',True, True);
   KADaoTable1.AccessExportToExcel('d:\tests\Table1.xls','Tab1',3,True, True);
   KADaoTable1.AccessExportToParadox('d:\tests\Table1.db',7,True, True);
   KADaoTable1.AccessExportToDBase('d:\tests\Table1.dbf',5,True, True);
   KADaoTable1.Close;
   KADaoTable1.Free;
 End;
```

Property IndexDefs : TKADaoIndexDefs;

Contains all Index definitions in the Table.
TKADaoIndexDefs is a descendant of TIndexDefs.
You can use Add method to create indexes at runtime.

Note: If Table is active then it is temporary closed and reopened after index was created.
Example:

```
KADaoTable1.IndexDefs.Add('Int','Int;Str',[ixPrimary]);
```

Also IndexDefs Property has an additional method called DeleteIndex(const Name : string):Boolean which deletes an index already stored in the IndexDefs. If Index exists in the table it is also deleted from the table.

Example:

```
KADaoTable1.IndexDefs.DeleteIndex('PrimaryKey');
```

Procedure CreateTable;

Same as TTable.CreateTable
Creates a new table based on the data sored in FieldDefs and IndexDefs properties
Example:

```
    KADaoTable1.Close;
KADaoTable1.TableName:='NewTestTable';
KADaoTable1.FieldDefs.Clear;
KADaoTable1.IndexDefs.Clear;
KADaoTable1.FieldDefs.Add('Int',ftInteger,0,False);
KADaoTable1.FieldDefs.Add('Str',ftString,10,False);
KADaoTable1.IndexDefs.Add('Int','Int;Str',[ixPrimary]);
KADaoTable1.CreateTable;
```

Procedure AppendTable;

Adds fields and indexes to an existing table based on the data sored in FieldDefs and IndexDefs properties

Example:

```
    KADaoTable1.Close;
KADaoTable1.TableName:='NewTestTable';
KADaoTable1.FieldDefs.Clear;
KADaoTable1.IndexDefs.Clear;
KADaoTable1.FieldDefs.Add('Int',ftInteger,0,False);
KADaoTable1.FieldDefs.Add('Str',ftString,10,False);
KADaoTable1.IndexDefs.Add('Int','Int;Str',[ixUnique]);
KADaoTable1.AppendTable;
```

Property TrackTransactions:Boolean;

True by Default

This property controls the way on which transactions works

If set to True KADaoDatabase will try to position CurrentRecord of all assigned tables to the position before transaction if a Rollback is called

If set to False CurrentRecord is always set to the first record of the table after Rollback

Example:

```
//*****************************************************************************
KADaoDatabase1.TrackTransactions:=True;
//*****************************************************************************
```

KADaoDBEngine

This component encapsulates all functions and properties of DAO DBEngine object.
It is a Delhi interface to the MS DAO.DBEngine COM object.
Its main purpouse is to use DAO withouth Delphi Dataset interfaces provided by KAdaoDatabase and KAdaoTable
All properties are with the same names as these in DAO.DBEngine COM object
All methods are with same names with thouse in KADaoDatabase
So you need to read MS DAO Help and TKADaoDatabse component help to get all needed information

KADaoWorkspace

This component encapsulates all functions and properties of DAO Workspace object.
It is a Delphi interface to the MS DAO.Workspace COM object.
Its main purpouse is to use DAO withouth Delphi Dataset interfaces provided by
KAdaoDatabase and KAdaoTable
All properties are with the same names as these in DAO.Workspace COM object
All methods are with same names with thouse in KADaoDatabase
So you need to read MS DAO Help and TKADaoDatabse component help to get all needed information

Property Active : Boolean;

Set this property to True to activate KADaoConnectionCheck component

Example:

```
KADaoConnectionCheck1.Database       := KADaoDatabase1;
KADaoConnectionCheck1.Interval       := 5000;
KADaoConnectionCheck1.ExceptionText  := 'Network Database %s is not available!';
KADaoConnectionCheck1.ErrorAction    := [RaiseException];
KADaoConnectionCheck1.Active         := True;
```

Property Interval : Integer;

This property determines the time interval on wich a checking for connection is performed.
Default is 1000 ms = 1 sec.
Do no set to more than 60000.
Example:

```
KADaoConnectionCheck1.Database       := KADaoDatabase1;
KADaoConnectionCheck1.Interval       := 5000;
KADaoConnectionCheck1.ExceptionText  := 'Network Database %s is not available!';
KADaoConnectionCheck1.ErrorAction    := [RaiseException];
KADaoConnectionCheck1.Active         := True;
```

Property Database : TKADaoDatabase;

This property determines KADaoDatabase component which will be monitored.
Example:

```
KADaoConnectionCheck1.Database       := KADaoDatabase1;
KADaoConnectionCheck1.Interval       := 5000;
KADaoConnectionCheck1.ExceptionText  := 'Network Database %s is not available!';
KADaoConnectionCheck1.ErrorAction    := [RaiseException];
KADaoConnectionCheck1.Active         := True;
```

Property ErrorCode : Integer;

This ReadOnly property contains last error code when check is performed
Zero means no error.
You can examine its value when a OnNoConnection event is triggered.
Example:

```
KADaoConnectionCheck1.Database       := KADaoDatabase1;
KADaoConnectionCheck1.Interval       := 5000;
KADaoConnectionCheck1.ExceptionText  := 'Network Database %s is not available!';
KADaoConnectionCheck1.ErrorAction    := [RaiseException];
KADaoConnectionCheck1.OnNoConnection := KADaoConnectionCheck1NoConnection;
KADaoConnectionCheck1.Active         := True;

procedure TForm1.KADaoConnectionCheck1NoConnection(Sender: TObject);
begin
   ShowMessage('No Connection to database '+
               (Sender As TKADaoConnectionCheck).Database.Database
                +#13#10
                +'Error code was: '+IntToStr(KADaoConnectionCheck1.ErrorCode));
end;
```

Property ExceptionText : String;

This property contains the text of the execption to be raised if **ErrorAction** property is set to **RaiseException**.

Example:

```
KADaoConnectionCheck1.Database       := KADaoDatabase1;
KADaoConnectionCheck1.Interval       := 5000;
KADaoConnectionCheck1.ExceptionText  := 'Network Database %s is not available!';
KADaoConnectionCheck1.ErrorAction    := [RaiseException];
KADaoConnectionCheck1.OnNoConnection := KADaoConnectionCheck1NoConnection;
KADaoConnectionCheck1.Active         := True;
```

Property ErrorAction : TErrActionSet;

This property contains the action to be performed when connection to Database is lost.
Can be combination

CloseDatabase - database is closed (which may raise an additional exception from DAO)

RaiseException - component raises an exception with the text contained in ExceptionText property.

Example:

```
KADaoConnectionCheck1.Database       := KADaoDatabase1;
KADaoConnectionCheck1.Interval       := 5000;
KADaoConnectionCheck1.ExceptionText  := 'Network Database %s is not available!';
KADaoConnectionCheck1.ErrorAction    := [RaiseException];
KADaoConnectionCheck1.OnNoConnection := KADaoConnectionCheck1NoConnection;
KADaoConnectionCheck1.Active         := True;
```

Event OnNoConnection : TNotifyEvent;

This event is triggered when connection to Database is lost.

Example:

```
KADaoConnectionCheck1.Database       := KADaoDatabase1;
KADaoConnectionCheck1.Interval       := 5000;
KADaoConnectionCheck1.ExceptionText  := 'Network Database %s is not available!';
KADaoConnectionCheck1.ErrorAction    := [RaiseException];
KADaoConnectionCheck1.OnNoConnection := KADaoConnectionCheck1NoConnection;
KADaoConnectionCheck1.Active         := True;

procedure TForm1.KADaoConnectionCheck1NoConnection(Sender: TObject);
begin
   ShowMessage('No Connection to database '+
               (Sender As TKADaoConnectionCheck).Database.Database
                +#13#10
                +'Error code was: '+IntToStr(KADaoConnectionCheck1.ErrorCode));
end;
```

Event OnConnectionRestored : TNotifyEvent;

This event is triggered when connection to Database is restored after some time.
This can give you the oportunity to reopen databases and tables.

Example:

```
KADaoConnectionCheck1.Database       := KADaoDatabase1;
KADaoConnectionCheck1.Interval       := 5000;
KADaoConnectionCheck1.ExceptionText  := 'Network Database %s is not available!';
KADaoConnectionCheck1.ErrorAction    := [RaiseException];
KADaoConnectionCheck1.OnConnectionRestored :=
KADaoConnectionCheck1ConnectionRestored;
KADaoConnectionCheck1.Active         := True;

procedure TForm1.KADaoConnectionCheck1OnConnectionRestored(Sender: TObject);
begin
 KADaoConnectionCheck1.Database.Open;
end;
```

KADaoConnectionCheck

Sometimes if your database files are on a network drive and this drive is gone for mome reason - i.e server shutdown or volume dismount the connection to your database is lost.
MSDao unfortenatelly does not have any rountines or events to inform the host application about such lost connection.

KADaoConnectionCheck component mades possible to monitor existance of the network drive and if it is gone to perform some actions like closing the database or fiering the exception or triggering an event
Also if after some time network connection to this drive is restored KADaoConnectionCheck component can trigger an event to inform the host application that volume is online again

You can use this to reopen database and tables or do something else.
Each KADaoDatabase component must have corresponding KADaoConnectionCheck component.

Property SmartOpen : Boolean;

True by default

When SmartOpen is True KADAODatabase first try to find **mdb** file with the filename specifyed in design time.

If filename does not exists KADAODatabase tryes to find same file in the program's startup folder.

Example:

```
   KADaoDatabase1.SmartOpen:=False;
KADaoDatabase1.Database   := 'c:\docs\runtime.mdb';
{*****************************************************************}
{if c:\docs\runtime.mdb does not exists}
{Datebase property will be changet to c:\exefilefolder\runtime.mdb}
{*****************************************************************}
KADaoDatabase1.Open;
{do some stuff ...}
KADaoDatabase1.Close;
```

Property CacheLookups : Boolean;

False by default
When CacheLookups is True and Table contains lookup fields then all such fields are cached from the lookup table
I.E each **Field.LookupCache** is set to True if **Field.FieldKind** is fkLookup
This can increase speed by 500%
If Lookup table information changes you must call RefreshLookups method to refresh all lookup fields.
Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.CacheLookups   := True;
KADaoTable1.Open;
{do some stuff ...}
{lookup table data has changed}
KADaoTable1.RefreshLookups;
{do some stuff ...}
KADaoTable1.Close;
```

Procedure RefreshLookups: Boolean;

When CacheLookups is True and Table contains lookup fields then all such fields are cached from the lookup table
I.E each **Field.LookupCache** is set to True if **Field.FieldKind** is fkLookup
This can increase speed by 500%
If Lookup table information changes you must call **RefreshLookups** method to refresh all lookup fields.
Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.CacheLookups   := True;
KADaoTable1.Open;
{do some stuff ...}
{lookup table data has changed}
KADaoTable1.RefreshLookups;
{do some stuff ...}
KADaoTable1.Close;
```

Property ExportMethod : TExportMethod;

This property controls the way of exporting data when one of
AccessExportToXXX methods is used.
Possible values: VisibleFields, AllFields
If ExportMethod is VisibleFields then only fields selected in the table editor and visibe will be exported
Otherwise all fields from the table will be exported
Default value is: VisibleFields
Example:

```
{This will export all fields of the Table/QueryDef/Query}
KADaoTable1.ExportMethod:=AllFields;
   KADaoTable1.AccessExportToMDB('C:\temp\test.mdb','table all' True, True);
{This will export ONLY VISIBLE/AVAILABLE fields of the Table/QueryDef/Query}
KADaoTable1.ExportMethod:=VisibleFields;
   KADaoTable1.AccessExportToMDB('C:\temp\test.mdb','table all' True, True);
```

Property DatabaseVersion : String;

This property gives information for the version of DAO used to create opened database.
For example you must use DAO 3.6 to open 3.51 database
In this case:

Version property = 3.6
DatabaseVersion property = 3.5

DatabaseVersion property will be different for Excel, Paradox and DBase files depending of version of the creator software

Property DatabaseParameters : String;

This property can contain some additional information for opening an database.
For example when openning an Excel file
DatabaseParameters can contain "HDR=NO; IMEX=1;"

Function GetDAOEnginesInstalled:TStringList;

This method returns all installed dao versions on the target computer.

Property RefreshSorted: Boolean;

False by default
When RefreshSorted is True and Table has SorttedBy property set to some value then after each Edit and Insert/Append
entire table will be refreshed so the new/modifyed record come in the righ place
However after this cursor is always positioned at FIRST row in the table and all bookmarks you have retrieved are lost.
Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.RefreshSorted  := True;
KADaoTable1.Open;
{do some stuff ...}
{lookup table data has changed}
KADaoTable1.RefreshLookups;
{do some stuff ...}
KADaoTable1.Close;
```

Procedure SetLockEdits(LockEdits : Boolean);;

Changes Locking type at runtime
If LockEdits is True then Pessimisting locking is set and 2K page of table containing the record is locked.
If LockEdits is False then Optimisting locking is set and 2K page of table containing the record is NOT locked.
Example:

```
KADaoTable1.Database   := KADaoDatabase1;
KADaoTable1.TableName  := 'Table1';
KADaoTable1.LockType=dbOptimistic;
KADaoTable1.Open;
KADaoTable1.SetLockEdits(True);
{Now a PESSIMISTIC LOCKING is active ...}
{do some stuff ...}
KADaoTable1.SetLockEdits(False);
{back to OPTIMISTIC LOCKING ...}
{do some stuff ...}
KADaoTable1.Close;
```

Property AutoFindIndex: Boolean;

True by default
When AutoFindIndex is True and TableType is StandardTable then when Locate procedure is used KADao tryes to find the best matchin index to perform seek command which is used by Locate.

Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.TableType      := dbOpenTable;
KADaoTable1.AutoFindIndex  := True;
KADaoTable1.Open;
KADaoTable1.Locate('FirstName;LastName',VarArrayOf(['Nancy','Davolio']),[]);
{ Do some stuff ...}
KADaoTable1.Close;
```

Property ShowGUID: Boolean;

False by default

When ShowGUID is TrueTable contains MS GUID fields then they are displayed in DB Aware controls

Otherwise a (BYTES) text will be displayed

Example:

```
KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.ShowGUID       := True;
KADaoTable1.Open;
{ Do some stuff ...}
KADaoTable1.Close;
```

Function GetGUIDAsString(GUID : String):String;

By getting GUID field as string you get the GUID in its BINARY form stored in the string
To convert GUID to a HUMAN READABLE Text Representation you mustt use this function

Example:

```
Var
 BGUID : String;
 SGUID : String;

KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.ShowGUID       := True;
KADaoTable1.Open;
KADaoTable1.First;
BGUID := KADaoTable1.FieldByName('GUID Filed').AsString; // Here BGUID will
contain a mess of chars
SGUID := KADaoTable1.GetGUIDAsString(BGUID); // Here SGUID will contain
something like '{B7FC3591-8CE7-11CF-9754-00AA00C00900}'
{ Do some stuff ...}
KADaoTable1.Close;
```

Function GetStringAsGUID(GUID : String) : TGUID;

This function will return a TGUID result
GUID parameter is a text representation of TGUID;
Example:

```
Var
 GUID   : TGUID;
 SGUID : String;
SGUID := '{B7FC3591-8CE7-11CF-9754-00AA00C00900}';
GUID := KADaoTable1.GetStringAsGUID(SGUID);
```

Function PutGUIDInString(GUID : String):String;

This function does the opposite of GetGUIDAsString - it converts text representation of TGUID in a binary form and puts it into result string

Example:

```
Var
 BGUID : String;
 SGUID : String;

KADaoTable1.Database       := KADaoDatabase1;
KADaoTable1.TableName      := 'Table1';
KADaoTable1.ShowGUID       := True;
KADaoTable1.Open;
KADaoTable1.First;
BGUID := KADaoTable1.FieldByName('GUID Filed').AsString; // Here BGUID will
contain a mess of chars
SGUID := KADaoTable1.GetGUIDAsString(BGUID); // Here SGUID will contain
something like '{B7FC3591-8CE7-11CF-9754-00AA00C00900}'
// do something with SGUID ...
BGUID := KADaoTable1.PutGUIDInString(SGUID );// Here BGUID AGAIN will contain a
mess of chars
KADaoTable1.Edit;
KADaoTable1.FieldByName('GUID Filed').AsString := BGUID;
KADaoTable1.Post;
{ Do some stuff ...}
KADaoTable1.Close;
```

Procedure RefreshDatasets;

Refreshes all datasets associated with the KADaoDatabase component without disconnecting from the DAO database.
Call RefreshDataSets to obtain latest contents of the tables and queries.
Example:

```
KADaoDatabase1.Database   := 'runtime.mdb';
KADaoDatabase1.Open;
KADaoTable1.TableName  := 'Table1';
KADaoTable2.TableName  := 'Table2';
KADaoTable2.TableName  := 'Table3';
KADaoTable1.Open;
KADaoTable2.Open;
KADaoTable3.Open;
KADaoDatabase1.RefreshDatasets;
```

Property QueryDefSQLModify: Boolean;

False by default

When QueryDefSQLModify is True any modifications maded to QueryDefSQLText are written back to the database.

Example:

```
KADaoTable1.Database             := KADaoDatabase1;
KADaoTable1.TableName            := 'Table1';
KADaoTable1.QueryDefSQLModify    := True;
KADaoTable1.QueryDefName         := 'Catalog';
KADaoTable1.QueryDefSQLText.Text := 'SELECT DISTINCTROW Categories.CategoryName,
'+
                                    'Categories.Description FROM Categories;'
KADaoTable1.QueryDefSQLText      := KADaoTable1.QueryDefSQLText;
{NOW 'Catalog' contains the new SQL text}
```

Procedure AccessExportToFoxPro(FileName :String; FoxProVersion:Integer; IncludeBlobs, DeleteOld:Boolean);

This method exports the table to the FoxPro Table

Warning! This routine works only with DAO 3.5 - in DAO 3.6 FoxPro is not supported!
Table must be opened before you can use this method!

FileName is the full path to the folder to which table will be exported.
InclubeBlobs when True causes method to export BLOB fields too otherwise BLOB fields are not exported;
DeleteOld when True deletes an existing file with the same name if a such file exists. Otherwise if file exists and DeleteOld is False an eception is raised.
FoxProVersion is an integer describing the version of FoxPro Table to be created.

20 'FoxPro 2.0
25 'FoxPro 2.5'
26 'FoxPro 2.6'
30 'FoxPro 3.0'

Limitations:
All Table rows is exported no matter what filters based on OnFilter event are applyed.
However filters based on Filter property are applyed to the result table
Tables based on parametrized QueryDefs are not exported.
Only those fields that are incliuded with fields editor are exported.

Example:

```
      Var
        KADaoTable1     : TKADaoTable;
       Begin
        KADaoTable1            := TKADaoTable.Create(Self);
        {set properties such Database etc ...}
        {...}
        KADaoTable1.TableName  := 'Table1';
        KADaoTable1.Open;
        KADaoTable1.AccessExportToTXT('d:\tests\Table1.txt',True, True);
        KADaoTable1.AccessExportToHTML('d:\tests\Table1.htm',True, True);
        KADaoTable1.AccessExportToExcel('d:\tests\Table1.xls','Tab1',3,True, True);
        KADaoTable1.AccessExportToParadox('d:\tests\Table1.db',7,True, True);
        KADaoTable1.AccessExportToFoxPro('d:\tests\Table1.db',30,True, True);
        KADaoTable1.AccessExportToMDB('d:\tests\demo.mdb','NewTable',True, True);
        KADaoTable1.Close;
        KADaoTable1.Free;
End;
```