Contents of the QDB units

Components
TQDB
TQDBItem
TQDBView
TQDBNavigator

TQDB provides basic, low-level, unstructured file access. TQDBItem inherits all from TQDB and adds structured access to individual fields in each item. TQDBView in turn inherits from TQDBItem and adds visual design and display of data. TQDBNavigator works with all three components.

Routines
RenameOrMoveFile
TempFileName

Global Variables
QDBTempFileLocation

Getting Started
Using TQDB for the first time
Using TQDBView for the first time
Converting from TQDBPanel to TQDBView

About the TQDB component

Purpose

The QDB components offer you a way to store data in a proprietary format without, at one extreme, having to construct the file-management from scratch or, at the other, using the Borland Database Engine with its various overheads.

The QDB components provide flat-file database storage for variable-sized items of data accessible *via* a single alphabetical, in-memory index. TQDB, itself, handles the general, low-level manipulation of such data. TQDBItem and TQDBView build upon TQDB to offer simpler, more automatic, and in the case of TQDBView visual, access to QDB data.

Database Structure

QDB files do not have a structure! It is very important to realize this—until you do you'll be wondering how to create QDB files or define what fields they have. They don't have fields: each item is a free-form block of bytes which you, the programmer, can decide to interpret anyway you please. QDB thus offers you the ability to define your own proprietary database format uniquely adapted to your needs. The demonstration programs supplied with QDB show several ways of parceling up program data into a QDB item.

(**Note**, however, that the TQDBView component *can* be used to automatically provide a structure to a QDB file based on the layout of ordinary Delphi controls on a panel.)

Accessing QDB Files

To use the TQDB component just drop it onto a form. Alternatively you can Create it on the fly and Free it when necessary. When you set the FileName property to the name of an existing QDB file that file is opened for access. Setting FileName to a name that does not exist creates a new QDB file. Setting FileName to nothing (*i.e.* to '') closes an open QDB file (saving its contents automatically if SaveOnClose is set). When a QDB file is open the Ready property is true. If you attempt to use a TQDB component when it is not Ready it will raise an exception. To protect a QDB file from accidental changes set its ReadOnly property.

Getting Around

You can navigate through the file with the FirstItem, LastItem, NextItem, & PrevItem methods. These methods navigate the list of keys (i.e., the index) which is held in memory and are usually very efficient. If the Filter property is set the navigation methods only access keys that match the filter pattern (which can slow things down considerably). The filtering mechanism can always be circumvented, however, by setting the ItemIndex property directly.

An Item can also be located *via* its key. KeyExists, ExactMatch and CloseMatch perform a fast binary search of the index. PartialMatch & PatternMatch perform a more methodical check of each key from the next item onward until a match is found with the start of the key or with a filter pattern respectively. A choice of pattern matching protocols is available.

Access to Items

Once an item has been selected (*i.e.*, been made the current item) it can be manipulated in many ways. The Key property holds the 'name' of the current item. The Get method retrieves the data as a Delphi memory stream. Memory streams are convenient since they handle huge blocks of memory transparently in all versions of Delphi. (ItemSize, GetItem, & GetStreamItem are only provided for compatibility and should be avoided) An internal cache of items (see CacheSize) attempts to speed up repeated access to the same items. Delete and Change operate on the current item. Items can be added to the QDB file using Add (AddItem, AddStreamItem, ChangeItem, ChangeStreamItem, & DeleteItem are provided for compatibility but should be avoided). You should sandwich batches of repeated operations between BeginUpdate and EndUpdate to enhance performance (see also PrepareToAdd).

An alternative way to access items is *via* the CurrentItem, Items, & ItemsByKey properties. These treat items as strings. Since Delphi strings know their own length and may contain any characters (*including* nulls) they can be used to handle any kind of data.

Housekeeping
Count contains the number of items in a QDB file. AssignKeyList generates a list (which may be filtered) of all the keys. Save commits the in-memory list of keys to disk. SaveAs creates a new QDB file with a new name. Pack tidies up the QDB file on disk, removing any defunct records. Kill (to be used with obvious care!) closes and deletes the QDB file.

Putting it Together
Integration of these various activities is achieved using TQDB's many events. In particular, the OnNavigate event is often an appropriate place to do any updating of a screen display.

In addition there are warning events which occur if operations are attempted on an empty or protected file or if the ItemIndex is set out of bounds. Warning events are, however, only triggered if event handlers have been provided. If no handler is provided for these events an exception is raised instead. An empty warning-event handler can be used to suppress such exceptions.

QDBNavigator
Navigation around the QDB file can be automated with the TQDBNavigator visual control which provides vcr-style buttons for the four navigation methods(FirstItem, LastItem, NextItem, and PrevItem) and other buttons to trigger events which can be handled in any way you like.

Multi-User Awareness
A QDB file can be opened by any number of users at once. Each user gets a *private* snapshot of the file as it was when opened. If a TQDB component detects that a QDB file has been opened by another user in the meantime it is careful not to overwrite the file without permission. Actions that would overwrite the file trigger a BeforeOverWrite event which can be handled to prevent or permit the overwrite.

Administrative Information
Often you'll want to store items in a QDB file which don't really belong with the regular data but are, for want of a better word, *administrative*. The case-sensitivity of a QDB file is such an "admin" item. You might want to store information about users, about dates and times of access, or about field or record structures—anything that should be stored with a particular file but kept apart from its data items.

Admin items can be stored and retrieved in the form of booleans, integers, or strings via the AdminAsBoolean, AdminAsInteger, and AdminAsString array properties. The array index is a string of type TKey. The number of admin items is given by AdminCount. A particular admin item can be removed by AdminDelete and a group of items (or all of them) by AdminClear. You can check for a particular admin item with AdminKeyExists or obtain a list of admin keys *via* AdminKeys.

Specifications
QDB files have the following characteristics:

- data items can be any size and contain any kind of information
- items are not structured into fields
- only one index is permitted
- the index is held in memory which makes for speed but limits the number of items
- the key to the index is a Delphi string up to 255 characters in length
- keys are sorted in ascending alphabetic order
- duplicate keys are not allowed
- multi-user awareness is limited
- does not work with Delphi data-aware controls (but see TQDBView)

About the TQDBItem component

Purpose

TQDBItem and TQDBView build upon TQDB by defining a field structure for QDB files.

Tasks

TQDBItem is meant primarily as an adjunct to TQDBView but it can be used separately. TQDBItem provides access to a structured QDB file field by field without the overhead of displaying the retrieved data in a panel. It may be easier to understand TQDBItem having first mastered TQDBView.

File Structure

A structured QDB file's structural information is stored as administrative data within the file according to a simple but flexible scheme. Files structured by QDBItem or QDBView thus form a special subset of QDB files: any QDBItem file can be accessed by a TQDB component but only some QDB files are accessible to TQDBItem. All valid QDBItem files are also valid QDBView files.

The primary way to define a file structure is *via* TQDBView which does the job visually. Changing a file's structure (by adding, removing, or renaming fields) is also best done using TQDBView's component editor. It is possible, though, to use TQDBItem's methods to do the job.

The structural information of a QDB file will usually be loaded automatically by TQDBView but can also be loaded into memory by FetchStructure. ClearStructure erases the structure both in memory and in the file on disk. Finally, a structure in memory can be written to a file by StoreStructure. The in-memory structure can be independently constructed, field-by-field, using AddField.

The file structure is accessed through the FieldCount, FieldNames, and FieldTypes properties. The FieldIndex method looks up a field index by name.

Access to Individual Fields

Before a field can be accessed the item must have been loaded from file and parsed into fields by the Fetch method. This happens behind the scenes so you should rarely need to call Fetch directly. Fetch is called implicitly by the FirstItem, PrevItem, NextItem, LastItem, Refresh, Edit, Insert, and Post methods which are the primary tools you should use to operate on a QDB file *via* TQDBItem..

Once stationed at the item of your choice the contents of its fields are available *via* the AsBoolean, AsInteger, AsDateTime, AsReal, and AsString properties. Lower-level access is also possible through the GetField method but is not recommended for general use.

About the TQDBView component

Purpose

TQDBView (like TQDBItem) builds upon TQDB by defining a field structure for QDB files.

Tasks

TQDBView defines a file structure *visually* within the Delphi IDE as you place controls (*ordinary* controls rather than data-aware ones) upon an associated panel. The contents of such controls are automatically stored and retrieved in a QDB file.

A TQDBView has an associated Panel. The contents of the panel's controls are stored in the QDB file. While FileName is blank or the assigned file is empty you can place controls upon the panel to define the file structure. Controls can be nested within other controls and all their data will be stored automatically. If you wish to exclude a particular control (or set of controls) you should set its Tag property to match the TQDBView's ExcludeTag property. If you just intend to view data already in the file that's all you have to do but if you intend to add more items you must also provide a handler for the OnKey event to supply a key by which to index each item.

You can navigate around a QDB file using TQDBNavigator or by calling the FirstItem, PrevItem, NextItem, LastItem, Refresh, Edit, Insert, and Post methods. The methods of the underlying TQDBItem and TQDB classes are also available.

Use ActiveColor and InactiveColor to select the way to display data which is being edited or just viewed respectively.

As supplied TQDBView knows how to handle controls descended from TCustomEdit, TRichEdit, TCustomRadioGroup, TCustomCheckBox, TCustomComboBox, TCustomListBox, and TImage. Edit controls (including memos) store the text they contain. Rich edit boxes store their formatted contents (just as SaveToFile would generate). Radio groups store the index of the selected item; check boxes their state; and combo boxes the selected text. List boxes store only the indices of selected items rather than the list itself. Finally image controls store the image data (internally prefixed with a code to identify the image format).

If you need to provide different behaviors for a kind of control, add an entirely new control type, or specialize within a class (as TRichEdit does within TCustomEdit) you can easily do so with RegisterControl. If you add a graphic format to TImage you must also register that format with TQDBView through RegisterGraphicFormat.

About the TQDBNavigator component

Purpose
The QDBNavigator component works with the QDB components (TQDB, TQDBItem, & TQDBView) to provide push-button access to individual items.

Tasks
You can choose the orientation of the navigator, which buttons are available, and which QDB component it acts upon. If the operating system allows, the buttons can be flat rather than raised.

It is possible to assign a hint to each button or replace a button's default glyph.

When linked to a QDB component the navigator reflects the state of the component by enabling or disabling its buttons, *e.g.*, if the QDB is not Ready all buttons are grayed out. The navigator also responds to the BoF and EoF conditions.

When any one of the navigators buttons is pressed three things happen in sequence:

first the BeforeAction event is triggered

then if there is a handler assigned to that button the corresponding event is triggered and if not the appropriate method of the associated QDB is called

finally the OnClick event is triggered

For example, if the associated QDB component is Q (of type TQDBView) and the post button is pressed and there is no handler for the OnPost event, Q's Post method is called to update the associated panel.

Using TQDB for the first time

(If you wish to use TQDBView's easier, higher-level approach you can skip to that section.)

The example projects (*address*, *animals*, and *archive*) show how to access QDB files directly *via* a TQDB component. TQDB treats each item in the file as an unstructured stream of bytes. It is up to the developer to parse the stream into meaningful values.

As such TQDB has no concept of fields or data types. If an item corresponds to a fixed-length data structure (*e.g.*, a Pascal record) it is easily read from the item stream in one step, *e.g.*:

```
type
  TX = record
    a: integer;
    b: extended;
    c: array [0..4] of char;
  end;
...
m:=TMemoryStream.Create;
try
  MyQDB.ExactMatch('aardvark');
  MyQDB.Get(m);
  // read the record in one go
  m.Read(MyX,SizeOf(TX));
finally
  m.Free;
end;
```

If the item contains sub-items of variable length the en/decoding must be more sophisticated. **Any** scheme can be used depending upon the developers needs, preferences, and ingenuity. One of the simplest approaches is to prefix each variable-length sub-item with an 32-bit integer code representing the length of the coming block of bytes.

```
type
  TX = record
    a: integer;
    b: extended;
    c: string;
  end;
...
m:=TMemoryStream.Create;
try
  MyQDB.ExactMatch('aardvark');
  MyQDB.Get(m);
  // read the record field by field
  m.Read(MyX.a,SizeOf(TX.a));
  m.Read(MyX.b,SizeOf(TX.b));
  m.Read(len,SizeOf(len));
  SetLength(MyX.c,len);
  m.Read(MyX.c[1],len);
finally
  m.Free;
end;
```

The TQDBItem and TQDBView components store each field prefixed with a 32-bit length code.

Using TQDBView for the first time

Starting from scratch is very simple.

- Place a TQDBView on a form. Call it Q.

- Place an ordinary panel (or similar container) on the form. Call it P.

- Add ordinary controls (not data-aware ones) to the panel in whatever way you wish. Standard edit controls, rich edit boxes, radio groups, check boxes, combo boxes, list boxes, and images are all recognized. In this case just add two edit boxes, Edit1 and Edit2, and two labels, Label1 and Label2. The text in the edit boxes will be stored but the labels will not.

- Set the FileName property of Q to create a new QDB file. This file will be automatically "branded" with the field structure defined by the controls on the panel.

- Provide a handler for the OnKey event. The simplest handler would just set the Key parameter to one of the fields, *e.g.*:

```
procedure TForm1.OnKeyHandler(Sender: TObject; var key: TKey);
begin
  key:=Edit1.Text;
end;
```

- Add a TQDBNavigator component to the form and set its QDB property to point to the QDBView, *i.e.*, Q..

- Compile and run the new application. You can view and modify the QDB file by pressing the various navigator buttons.

- For example, press the Insert button. You will see the edit boxes change color. Type some text into the boxes and press the Post button. Repeat this a few times.

- You can move forward and backward in the usual way or you can press Edit to change some values.

That's all there is to the basic use of TQDBView. Have a look at the example programs (vanimals and vaddress) which show slightly more complex use of TQDBView. Compare these to the versions (animals and address) which use TQDB directly. The new versions are considerably simpler but slightly slower and the QDB files they produce are less compact. The trade-off is a common one.

Conversion of Old Applications

The prototype of TQDBView, TQDBPanel, was a descendant of TPanel and had to be associated with a TQDB component to handle the actual QDB file. The situation has been reversed in the current release. TQDBView (and TQDBItem) descend from TQDB and, as such, handle their own QDB files. TQDBView is now a **non-visual** component with a Panel property connecting it to the panel it is to manage. The change in architecture now concentrates the database management in the TQDBView component instead of distributing it between TQDBPanel and TQDB. The shift makes TQDBView much more flexible than TQDBPanel but creates some complexity in converting existing applications from TQDBPanel to TQDBView. If you follow the steps below, however, it shouldn't be *too* difficult.

Before you do anything make a backup of you original files—all of them code and data. Make two! Be careful ...

- First—**before** loading the project in Delphi—use the provided **ConvQDB** utility on the units of your application. ConvQDB lets you choose a .pas file and then loads it and the associated .dfm file into its editor panes. You need to think carefully about the changes to make. In simple applications you will want to change every occurrence of TQDBPanel into TPanel and every TQDB into TQDBView. To help you do this (in both .pas and .dfm panes) I have provided a simple search button to locate the next instance of the string "TQDB". ConvQDB saves the original files with extensions .paq and .dfq so you can recover them if necessary.

- Once you have used ConvQDB you should be able to load the files into the Delphi IDE and complete the conversion by hand. On loading you will may get some error messages about unknown properties in the .dfm file but these should be harmless enough to ignore. For example, the TQDBPanel had a QDB property which is absent in the TPanel it has been transformed into.

- The remainder of the conversion has two parts: one mechanical and the other more careful. The mechanical process involves transferring database actions from objects which were TQDBPanels (and are now ordinary TPanels) to objects which were TQDBs (and are now TQDBViews).

- The more careful part requires checking your application's logic to make sure it does the same job with the different division of responsibilities between TQDBView and TPanel. It can be easy, for example, to forget to replace property values that were lost in the form conversion—check the backup copies of the old dfm files to jog your memory. Another common glitch occurs when a TQDBPanel had its Enabled property set false. In operation it would have been toggled as necessary as the database was accessed. Now it will just sit there making you wonder why you can't browse the file—remove it.

TQDB Component

Properties Methods Events Tasks

Unit
QDB

Description
TQDB was written in Borland Delphi by Robert R. Marsh, S.J.

This is Version 2.1.

Please let me know:

- of any bugs you discover
- what functionality you would like to see added
- how you might like to see something done differently

I can be contacted at rrm@sprynet.com. Check out my web site: http://home.sprynet.com/sprynet/rrm/

You may distribute TQDB as widely as you wish as long as you distribute the entire package intact. You may use TQDB without charge as long as you make no profit from its direct sale. If you do use this component you might consider making a gift to your favorite charity. You must also give me credit if you use TQDB in any projects of your own and maybe send me a copy of your work.

Users of TQDB must accept the following disclaimer of warranty:

TQDB is supplied as is. 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 TQDB.

Properties

▶ Run-time only
🔑 Key properties

In TQDB

AboutAuthor
AboutVersion
▶ AdminAsBoolean
▶ AdminAsInteger
▶ AdminAsString
AggressiveUpdate
▶
🔑 BoF
CacheFrequency
CacheSize
Compression
▶
🔑 Count
▶ CurrentItem
▶
🔑 EoF
ExpandedFileNames
🔑 FileName
Filter
▶ FilteredCount
▶
🔑 ItemIndex
▶ Items
▶ ItemsByKey
▶ ItemSize
▶
🔑 Key
▶ KeyCaseSensitive
▶ MatchWholeWord
▶ Password
🔑 ReadOnly
▶ Ready
SaveOnClose
UseGrepMatch

Methods

🔑 Key methods

In TQDB

- 🔑 Add
- AddItem
- AddStreamItem
- AdminClear
- AdminCount
- AdminDelete
- AdminKeyExists
- AdminKeys
- 🔑 AssignKeyList
- BeginUpdate
- CacheFlush
- CacheStatistics
- Cancel
- 🔑 Change
- ChangeItem
- ChangeStreamItem
- 🔑 CloseMatch
- Compress
- 🔑 Delete
- DeleteItem
- Edit
- EndUpdate
- 🔑 ExactMatch
- Expand
- 🔑 FirstItem
- 🔑 Get
- GetItem
- GetStreamItem
- Insert
- 🔑 KeyExists
- Kill
- 🔑 LastItem
- 🔑 NextItem
- OrphanToRecover
- Pack
- 🔑 PartialMatch
- PartialMatchInit
- 🔑 PatternMatch
- PatternMatchInit
- Post
- 🔑 PrepareToAdd
- 🔑 PrevItem
- Recover
- Refresh
- 🔑 Save
- 🔑 SaveAs
- SetMatchChars
- UpdateNavigator

Events

🔑 Key events

In TQDB

- BeforeKill
- BeforeOverWrite
- OnAdded
- OnChanged
- OnDeleted
- OnDemandPassword
- OnFileAssigned
- OnFound
- OnKilled
- 🔑 OnNavigate
    - ProgressUpdate
    - WarnNoData
    - WarnOutOfBounds
    - WarnReadOnly

AboutAuthor property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property AboutAuthor : string[40];
```

Description
A read-only property, visible in ObjectInspector, giving the author's name and e-mail address.

AboutVersion property

Applies to

TQDB, TQDBItem, TQDBView

Declaration

```
property AboutVersion : string[5];
```

Description

A read-only property giving the version of QDB.

AdminAsBoolean property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property AdminAsBoolean[Key : TKey] : boolean;
```

Description
Stores and retrieves Admin items as boolean values, e.g.:

```
      Q.AdminAsBoolean['one']:=true;
      ...
      if Q.AdminAsBoolean['one'] then ...
```

If an attempt is made to retrieve an item which doesn't exist an EQDBBadKey exception is raised.

Storing an item with an existing key overwrites the previous value.

If the key exists but the item is not boolean the result is undefined.

Run-time only

AdminAsInteger property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property AdminAsInteger[Key : TKey] : longint;
```

Description
Stores and retrieves Admin items as (long) integer values, e.g.:

```
        Q.AdminAsInteger['two']:=2;
        ...
        n:=Q.AdminAsInteger['two'];
```

If an attempt is made to retrieve an item which doesn't exist an EQDBBadKey exception is raised. Storing an item with an existing key overwrites the previous value.

If the key exists but the item is not integer the result is undefined.

Run-time only

AdminAsString property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property AdminAsString[Key : TKey] : string;
```

Description
Stores and retrieves Admin items as string values, e.g.:

```
      Q.AdminAsString['three']:='Robert R. Marsh, SJ';
      ...
      Edit1.Text:=Q.AdminAsString['three'];
```

If an attempt is made to retrieve an item which doesn't exist an EQDBBadKey exception is raised. Storing an item with an existing key overwrites the previous value.

If the key exists but the item is not a string the result is undefined.

Under D16 strings longer than 255 characters are truncated.

Run-time only

AggressiveUpdate property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property AggressiveUpdate : boolean;
```

Description
The FirstItem and LastItem methods always set BoF and EoF correctly but the overhead involved in getting PrevItem and NextItem to do the same may be prohibitive when a filter is in force. By default, *i.e.* when AggressiveUpdate is false, PrevItem and NextItem only set BoF and EoF when they have tried and failed to move to an earlier (or later) item. Note, therefore, that you can be at the first or last item without BoF/EoF being set correctly. This behavior is similar to that adopted by the BDE.

By setting AggressiveUpdate to true, however, PrevItem and NextItem go to the extra trouble of checking for BoF/EoF. In a large file with a restrictive filter this could prohibitive as it might involve scanning every key.

AggressiveUpdate does not affect the behavior of the other navigation methods since they, by default, do not update the BoF or EoF properties. If you are navigating around a QDB file in this way and you do want BoF/EoF to be maintained (and any attached navigator updated) you can call UpdateNavigator manually.

BoF property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property BoF : boolean;
```

Description
The behavior of the BoF and EoF properties depends upon the AggressiveUpdate property.

If AggressiveUpdate is true BoF is set whenever FirstItem or PrevItem moves to the first item in the file (taking into account any active Filter). Note, however, that if a filter is in effect there is an extra overhead when PrevItem checks for BoF. Since this could be considerable the default condition is for AggressiveUpdate to be false.

When AggressiveUpdate is false PrevItem only sets BoF when it tries and fails to move to an earlier item. Note, therefore, that you can be at the first item without BoF being true.

Setting ItemIndex directly does not affect BoF.

BoF and EoF both call UpdateNavigator to let an attached QDBNavigator component to change the status of its buttons accordingly.

Run-time only
Read-only

CacheFrequency property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property CacheFrequency : integer;
```

Description
The CacheFrequency property (which has been superceded by the CacheStatistics method) returns the percentage of item-requests that are serviced by the internal item-cache rather than by reading from the disk. Values range from 0 (when every item has been fetched fresh from the disk) to 100 (when all requests have been honored by the cache).

CacheSize can be adjusted until CacheFrequency hits some appropriately high value indicating high performance. Remember though that a very large cache both consumes memory and has its own performance overhead.

CacheFrequency is reset automatically whenever CacheSize is changed but can also be reset manually by assigning it the value zero, i.e.:

```
      CacheFrequency := 0;
```

N.B.: You really do need to check whether item-cacheing actually improves performance on your system. The default system disk-cache may be good enough to do away with any item-cacheing benefits.

CacheSize property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property CacheSize : longint;
```

Description
The CacheSize property sets the size of the internal cache that is used to speed up access to items of data. The default value is 128K. You should choose a CacheSize that balances accomodation of a significant number of data items against consumption of memory. The CacheFrequency property or CacheStatistics method can be used to measure how succesful the cache is.

The item-cache stores as many items as will fit in the allotted memory. When the cache gets full as many items as necessary are removed to accommodate the new item. The items discarded first are among those least recently used.

N.B. Caching has its own overhead. If QDB's temporary files are stored on a ram (or other fast) drive caching may actually degrade performance. You should always check to see if the item cache is helping.

Compression property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property Compression : boolean;
```

Description
If Compression is set to true every new or modified item is saved in a compressed form. If Compression is false every new or modified item is saved in uncompressed form. Other items are unaffected.

Compression is performed on an item-by-item basis and, as such, is more effective for large items. The compression algorithm, though rapid, achieves good compression ratios on such items (especially text or bitmap data) and doesn't swell small or incompressible items.

It is also possible to compress or expand the whole QDB file.

Count property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property Count : TItemIndex;
```

Description
The Count property gives the number of items in the QDB file. Admin items are not counted. See also FilteredCount.

If Count is zero, i.e. the file is empty, many operations trigger a WarnNoData event or a EQDBNoData exception..

Run-time only
Read-only

CurrentItem property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property CurrentItem : string;
```

Description
The CurrentItem property contains the current item as a Delphi string, *e.g.*:

```
      if Q.ExactMatch('Marsh') then Edit1.Text:=Q.CurrentItem;
```

Since Delphi strings can contain any character (even #0) they can be used to hold any kind of item:

```
      bmp:=TFileStream.Create('test.bmp',fmOpenReadWrite);
      SetString(s,bmp.Size);
      bmp.Read(s[1],Length(s));
      Q.CurrentItem:=s;
```

N.B. This technique is of limited use in D16 because of the 255 character limit on strings.

See also Items and ItemsByKey.

Run-time only

EoF property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property EoF : boolean;
```

Description
The behavior of the BoF and EoF properties depends upon the AggressiveUpdate property.

If AggressiveUpdate is true EoF is set whenever LastItem or NextItem moves to the last item in the file (taking into account any active Filter). Note, however, that if a filter is in effect there is an extra overhead when NextItem checks for EoF. Since this could be considerable the default condition is for AggressiveUpdate to be false.

When AggressiveUpdate is false NextItem only sets EoF when it tries and fails to move to a later item. Note, therefore, that you can be at the last item without EoF being true.

Setting ItemIndex directly does not affect EoF.

BoF and EoF both call UpdateNavigator to let an attached QDBNavigator component to change the status of its buttons accordingly.

Run-time only
Read-only

ExpandedFileNames property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property ExpandedFileNames : boolean;
```

Description
The ExpandedFileNames governs whether the value of the FileName property is expanded to a fully qualified path name based on the currently selected directory.

By default ExpandedFileNames is true and FileNames are expanded as they are entered. Otherwise, the FileName property is not expanded and the QDB file opening and closing routines will look for the specified file in the current directory. Such behavior affords a greater flexibility but can be problematic: *e.g.*, if the current directory is changed unexpectedly QDB may not be able to find the required file and then go ahead and create an empty one in the current directory leaving yuor application to comaplin about empty data files.

FileName property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property FileName : TQDBFileName;
```

Description
The FileName property contains the name of the physical QDB file. FileName is either empty (*i.e.* ''), in which case there is no open QDB file, or refers to the current open QDB file. The default extension is 'QDB'. If the ExpandedFileNames property is true (the default) a new FileName is expanded to its fully qualified form and stored that way. Otherwise the FileName refers to the current directory, increasing flexibility but also the risk of referring to the wrong QDB file.

Setting FileName opens or closes the QDB file. If FileName is cleared (*i.e.*, set to '') the file currently open is closed. If a non-null value is assigned there are two possible outcomes.

> if FileName refers to an existing QDB file, the file is opened and readied for use.
>
> if FileName does not refer to an existing file, a new QDB file is created and readied for use.

If SaveOnClose is true, closing a QDB file also saves any changes that have been made. Otherwise when FileName is changed any changes made the open QDB file will be lost.

FileName is also changed implicitly by the SaveAs method.

Filter property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property Filter : TKey;
```

Description
The Filter property effectively screens certain keys from the index. If Filter is null (i.e. '') navigation operations can access every key in the index but, if not, only keys which match the Filter pattern are accessible.

The algorithm for pattern-matching is governed by the setting of UseGrepMatch.

N.B.: Be aware that filtered navigation carries a much greater overhead than unfiltered, *e.g.*, with 10 character keys and the simple matching protocol:

filter type	relative time
none	20
match every key	80
match 1 in 10	240
match 1 in 100	1600
match 1 in 1000	18000

The overhead is even greater for grep-style matching.

Setting ItemIndex directly is not affected by the Filter, nor are the search methods.

FilteredCount property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property FilteredCount : TItemIndex;
```

Description
The FilteredCount property gives the number of items in the QDB file which match the current Filter. Admin items are not counted. Each access of the FilteredCount property performs a brute force scan of the index so extensive use of this property is not recommended.

The true number of items (ignoring any filter) is given by Count.

Run-time only
Read-only

Items property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property Items[itemindex : TItemIndex] : string;
```

Description
The Items array property offers access to the items of a QDB file as Delphi strings, e.g.:

```
      for n:=1 to Q.Count do
        ShowMessage(Q.Items[n-1]);
```

Since Delphi strings can contain any character (even #0) they can, in principle, be used to hold any kind of data, *e.g.*:

```
      bmp:=TFileStream.Create('test.bmp',fmOpenReadWrite);
      SetString(s,bmp.Size);
      bmp.Read(s[1],Length(s));
      Q.Items[Q.ItemIndex]:=s;
```

This technique is of limited use in D16 because of the 255 character limit on strings.

See also CurrentItem and ItemsByKey

Run-time only

ItemsByKey property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
`property ItemsByKey[Key : TKey] : string;`

Description
The ItemsByKey array property offers access by key to the items of a QDB file as Delphi strings, e.g.:

```
        ShowMessage(Q.ItemsByKey['Marsh']);
```

Or, since ItemsByKey is the default array property,

```
        ShowMessage(Q['Marsh']);
```

When assigning an item to this property if the key doesn't already exist the item is added. Otherwise the item is changed. When reading the value of this property if the key doesn't exist a EQDBBadKey exception is raised.

Since Delphi strings can contain any characters (even nulls) they can be used to hold any kind of item:

```
      bmp:=TFileStream.Create('test.bmp',fmOpenReadWrite);
      SetString(s,bmp.Size);
      bmp.Read(s[1],Length(s));
      Q['bmp']:=s;
```

This technique is of limited use in D16 because of the 255 character limit on strings.

See also CurrentItem and Items

Run-time only

ItemIndex property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property ItemIndex : TItemIndex;
```

Description
The ItemIndex property holds the location in the index of the current item. Setting ItemIndex permits direct navigation of the database. Its value lies between 0 and (Count -1) and can be assigned to any intermediate value irrespective of the setting of the Filter property. Setting ItemIndex to a new value triggers the OnNavigate event.

N.B.: The ItemIndex of a particular item can change as other items are added to or deleted from the file so, in most circumstances, other navigation methods may be more appropriate, *e.g.*, FirstItem, LastItem, NextItem, PrevItem, or the search methods CloseMatch, ExactMatch, PartialMatch and PatternMatch.

Run-time only

ItemSize property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property ItemSize : TDataIndex;
```

Description
The ItemSize property gives the size of the current item (in bytes). This makes it possible to allocate a suitable block of memory for the GetItem method.

N.B. In QDB 2.x ItemSize carries a *significant* overhead and should be avoided. Instead use the Get method which manages its own memory.

Run-time only
Read-only

Key property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property Key : TKey;
```

Description
The Key property contains the value of the current item's key.

Keys are kept in alphabetical order and may or may not be case-sensitive. Each key must be unique.

Run-time only
Read-only

KeyCaseSensitive property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property KeyCaseSensitive : boolean;
```

Description
Governs the case-sensitivity of the index.

KeyCaseSensitive = true	'Marsh' and 'marsh' are different keys
KeyCaseSensitive = false	'Marsh' and 'marsh' are duplicate keys

N.B. Also affects the various search and filter methods.

This property should not be changed frequently on an open file since, when it is changed, the index has to be re-checked for duplicates. In particular, when the index goes from being case-sensitive to case-insensitive there is the possibility of keys becoming non-unique. Any such keys are automatically purged from the database (which may not be what you want to do!). The purge operation also invokes the Pack method.

Run-time only

MatchWholeWord property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property MatchWholeWord : boolean;
```

Description
The MatchWholeWord property governs the behavior of the grep-style pattern-matching algorithm in force when the UseGrepMatch property is true.

Password property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
`property Password : TPassword;`

Description
Setting the Password property causes the QDB file to be encrypted when next closed. An encrypted file can only be opened if the appropriate Password is set (just) before assigning the FileName, e.g.:

```
      Q.PassWord:='tripe';
      Q.FileName:='trash.qdb';
```

If the wrong Password is given an EQDBInvalidPW exception is raised.

N.B. When a file is closed (*i.e.*, FileName is set to '') Password is also cleared to increase security.

See also: OnDemandPassword

Run-time only

ReadOnly property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property ReadOnly : boolean;
```

Description
ReadOnly can be set to prevent the QDB file being changed. If the physical file on disk is read-only the ReadOnly property is set automatically.

Attempting any action that would change the file generates either a EQDBReadOnly exception or, if a handler has been assigned for it, a WarnReadOnly event.

Ready property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property Ready : boolean;
```

Description
Ready is only true when a QDB file is open, i.e., a valid FileName has been assigned.

Attempting most actions when not Ready generates a EQDBNoFile exception.

Run-time only
Read-only

SaveOnClose property

Applies to

TQDB, TQDBItem, TQDBView

Declaration

```
property SaveOnClose : boolean;
```

Description

The SaveOnClose property governs whether QDB files are automatically saved when the FileName is changed or whether changes are discarded. By default SaveOnClose is true.

See also Save and SaveAs

UseGrepMatch property

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
property UseGrepMatch : boolean;
```

Description
Governs whether the Filter property and PatternMatch method use the simple pattern-matching protocol or a more complex grep-style pattern-matching.

Add method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Add(Stream : TStream; Key : TKey);
```

Description
The Add method inserts an item into the QDB file. The item is provided as a stream (usually, but not necessarily, a memory stream). The Key supplied is used to index the item. The index is kept in alphabetical order of key with or without case-sensitivity.

Add is to be preferred to AddItem and AddStreamItem which are provided only for compatibility.

If you intend to add many items at once use the PrepareToAdd method to speed up the process.

Attempting to add an item with a duplicate key will raise an EQDBIndexError exception. If the FileName property is not set (i.e., Ready is false) the Add method raises an EQDBNoFile exception.

AddItem method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure AddItem(ItemPtr : pointer; ItemLen : TDataIndex; Key : TKey);
```

Description
AddItem (provided primarily for compatibility) adds an item to the QDB file. The item is a block of data pointed to by the ItemPtr parameter with its length (in bytes) in the ItemLen parameter. The Key supplied is used to index the data. The index is kept in alphabetical order of key with or without case-sensitivity.

AddItem is converted internally into a call to Add which should *always* be used in preference since it handles huge items transparently under all versions of Delphi.

AddStreamItem method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
procedure AddStreamItem(Stream : TStream; Key : TKey);
```

Description

AddStreamItem (provided only for compatibility) is exactly equivalent to Add which should always be used in preference.

AdminClear method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure AdminClear(StartOfKey : TKey);
```

Description
The AdminClear method deletes all those Admin items from the QDB file whose keys begin with StartOfKey. If StartOfKey is '' (*i.e.*, an empty string) every Admin item is deleted.

AdminCount method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
function AdminCount : TItemIndex;
```

Description

The AdminCount method gives the number of Admin items in the file.

Note that, unlike Count, AdminCount is a method and not a property.

AdminDelete method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure AdminDelete(Key : TKey);
```

Description
The AdminDelete method removes the Admin item with the specified Key. If the key does not exist a EQDBBadKey exception is raised. AdminKeyExists can be used to check.

AdminKeyExists method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
function AdminKeyExists(Key : TKey) : boolean;
```

Description

Checks the Admin items to see if a particular Key is present.

AdminKeys method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
function AdminKeys(Keys : TStrings; StartOfKey : TKey) : longint;
```

Description

The AdminKeys method fills Keys with a list of all the Admin keys which begin with StartOfKey. If startofkey is empty ('') all the Admin keys are included. The return value corresponds to the number matching keys.

```
keys:=TStringList.Create;
Q.AdminKeys(keys,'QDB');
```

AssignKeyList method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure AssignKeyList(Keys : TStrings);
```

Description
The AssignKeyList method fills Keys with a list of all the keys in the index. If a Filter is in force, Keys contains only the matching keys.

AssignKeyList can be used to fill a list box or other control with the QDB keys. Each time AssignKeyList is called the list is generated afresh. During this process ProgressUpdate events are triggered at intervals.

BeginUpdate method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
procedure BeginUpdate;
```

Description

The BeginUpdate and EndUpdate methods speed repetitive tasks by eliminating triggering of the OnNavigate event.

Batches of, say, Add calls can be sandwiched between BeginUpdate and EndUpdate. Since the OnNavigate event is typically used to refresh a screen display, its elimination can speed things up considerably.

BeginUpdate and EndUpdate can be nested -- *i.e.*, it takes two EndUpdates to balance two BeginUpdates.

CacheFlush method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
procedure CacheFlush;
```

Description

Clears the item-cache.

CacheStatistics method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure CacheStatistics(var MaxSize, CurSize, CurLen, HitRatio, DropRatio :
longint);
```

Description
Gives performance information on the item-cache.

MaxSize	the upper limit on cache memory (bytes)
CurSize	the current size of the cache (bytes)
CurLen	the current number of items in the cache
HitRatio	the percentage of accesses that the cache can service
DropRatio	the percentage of accesses that force items out of the cache

You **do** need to check that the item-cache actually helps performance rather than hindering it! The cache imposes a small overhead, especially when old items have to be forced out of the cache to make way for new ones. If your hard disk is fast, or your system's disk-cacheing is efficient, the overhead might actually outweigh any performance benefits.

Cancel method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Cancel;
```

Description
Cancel belongs to the family of QDB methods corresponding to the buttons of TQDBNavigator. Cancel aborts the current operation, *e.g.,* abandons an Edit or Insert. Cancel can be called directly or *via* TQDBNavigator.

Cancel behaves differently in the different descendants of TQDB as each descendant adds its own functionality.

In TQDB
Does nothing more than switch out of edit or insert mode and call UpdateNavigator.

In TQDBItem
As above but additionally undoes any changes to the current item by reloading.

In TQDBView
As above but also puts the panel into the appropriate mode -- see AutoEdit.

Change method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

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

Description

The Change method replaces the current item with the item in the stream parameter. If the new item is smaller than the one replaced it simply overwrites it. Otherwise the old item is deleted and the new one added. Be aware that such repeated changes can cause the file to grow and be in need of packing..

A successful Change generates an OnChanged event.

Since it handles huge items transparently under all versions of Delphi, the Change method should be preferred to ChangeItem or ChangeStreamItem which are only provided for compatibility.

If the FileName property is not set (*i.e.*, Ready is false) the Change method raises an EQDBNoData exception.

ChangeItem method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure ChangeItem(ItemPtr : pointer; ItemLen : TDataIndex);
```

Description
The ChangeItem method (**provided only for compatibility**) replaces the current item with the item in the ItemPtr parameter. The ItemLen parameter indicates the length of the new item. If the new item is smaller than the one replaced it simply overwrites it. Otherwise the old item is deleted and the new one added. Be aware that such repeated changes can cause the file to grow and need to be Packed.

Since it handles huge items transparently under all versions of Delphi, the Change method should always be preferred.

ChangeStreamItem method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure ChangeStreamItem(Stream : TStream);
```

Description
ChangeStreamItem is exactly equivalent to Change which should be used instead.

Clear method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Clear;
```

Description
The Clear, Fetch, and Store methods operate at a lower level than the Cancel, Delete, Edit, Insert, Post, and Refresh methods and should generally be avoided unless necessary.

Clear blanks the current item in a way appropriate to the component.

In TQDBItem
Clears the current item and its fields.

In TQDBView
As above but also clears the associated Panel.

CloseMatch method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
function CloseMatch(Partialkey : TKey) : boolean;
```

Description
The CloseMatch method searches the list of keys for an alphabetical match with the PartialKey parameter:

If an *exact* match occurs ItemIndex is set to that item and the function returns true.

If the partial key only matches the *beginning* of a key ItemIndex is set to that item and the function returns false. In either case an OnFound event is triggered.

See also: ExactMatch, PartialMatch, & PatternMatch.

Compress method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Compress;
```

Description
The Compress method sets the Compression property to true and compresses every item in the QDB file.

During compression the ProgressUpdate event is triggered periodically. Compress involves an implicit pack of the QDB file.

See also: Expand.

Delete method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Delete;
```

Description
Delete belongs to the family of QDB methods corresponding to the buttons of TQDBNavigator. Delete erases the current item. Delete can be called directly or *via* TQDBNavigator.

Delete behaves differently in the different descendants of TQDB as each descendant adds its own functionality.

In TQDB
The Delete method deletes the current item irreversibly. ItemIndex shifts to the previous item if there is one. The QDB file on disk does not shrink until the Pack method has been used.

When successful Delete generates an OnDeleted event.

If the FileName property is not set (*i.e.*, Ready is false) the Delete method raises an EQDBNoFile exception.

In TQDBItem
As above but additionally loads the newly current item or, if the file is empty, creates an empty one.

In TQDBView
As above but also puts the panel into the appropriate mode -- see AutoEdit.

DeleteItem method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure DeleteItem;
```

Description
The DeleteItem method deletes the current item irreversibly. ItemIndex shifts toward the beginning of the file. The QDB file on disk does not shrink until the Pack method has been used.

When successful DeleteItem generates an OnDeleted event.

If the FileName property is not set *(i.e.*, Ready is false) the DeleteItem method raises an EQDBNoFile exception.

In TQDB
DeleteItem is exactly equivalent to Delete which should be used instead.

In TQDBItem & TQDBView
Delete behaves differently in TQDBItem and TQDBView but DeleteItem does not.

Edit method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Edit;
```

Description
Edit belongs to the family of QDB methods corresponding to the buttons of TQDBNavigator. Edit switches into edit mode. Edit can be called directly or *via* TQDBNavigator.

Edit behaves differently in the different descendants of TQDB as each descendant adds its own functionality.

In TQDB
Simply switches into edit mode and calls UpdateNavigator.

In TQDBItem
As above.

In TQDBView
As above but also puts the panel into editing mode.

EndUpdate method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure EndUpdate;
```

Description
The BeginUpdate and EndUpdate methods simplify repetitive tasks by eliminating triggering of the OnNavigate event.

Batches of, say, Add calls, can be sandwiched between BeginUpdate and EndUpdate. Since the OnNavigate event is typically used to refresh a screen display, its elimination can speed things up considerably.

BeginUpdate and EndUpdate can be nested -- *i.e.*, it takes two EndUpdates to balance two BeginUpdates.

ExactMatch method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
function ExactMatch(Key : TKey) : boolean;
```

Description
The ExactMatch method searches the list of keys for an alphabetical match with the Key parameter:

If (and only if) an exact match occurs ItemIndex is set to that item, the function returns true, and an OnFound event is triggered.

See also: CloseMatch, PartialMatch, & PatternMatch.

Expand method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Expand;
```

Description
The Expand method sets the Compression property to false and decompresses every item in the QDB file.

During decompression the ProgressUpdate event is triggered periodically. This operation also packs the file.

See also: Compress.

FirstItem method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure FirstItem;
```

Description

In TQDB
The FirstItem method moves to the first item in the index. If a Filter pattern has been set FirstItem finds the first matching item.

A call to FirstItem, when it actually moves to a different item, generates an OnNavigate event.

See also: BoF.

In TQDBItem
As above but also loads the first item and parses it into fields.

In TQDBView
As above but also displays the first item in the Panel.

Get method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

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

Description

The Get method loads the current item into the Stream parameter. If the FileName property is not set (*i.e.*, Ready is false) the Get method raises an exception.

GetItem method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
procedure GetItem(ItemPtr : pointer);
```

Description

The GetItem method (**provided only for compatibility**) loads the current item into the memory referenced by ItemPtr. It is essential that the memory referenced by ItemPtr is large enough to hold the item. The ItemSize property can be used to discover how much memory is needed.

GetItem has been superceded by the Get method which handles memory allocation itself using memory streams. Only use GetItem and ItemSize if you absolutely have to.

If the FileName property is not set the GetItem method raises an exception.

GetStreamItem method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

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

Description

GetStreamItem is exactly equivalent to Get which should be used instead.

GrepMatches method

Applies to
TQDB

Declaration
```
function GrepMatches(key: TKey; pattern: TKey): boolean;
```

Description
The GrepMatches method implements the grep-style QDB pattern-matching protocol. It is a protected method that cannot be called directly by your programs but can be overridden if you wish to implement a more sophisticated scheme.

Be aware, however, that pattern-matching *via* the Filter property already incurs quite an overhead -- one you may not want to increase with a complex matching routine.

Insert method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
`procedure Insert;`

Description
Insert belongs to the family of QDB methods corresponding to the buttons of TQDBNavigator. Insert switches into insert mode. Insert can be called directly or *via* TQDBNavigator.

Insert behaves differently in the different descendants of TQDB as each descendant adds its own functionality.

In TQDB
Does nothing more than switch into insert mode and call UpdateNavigator.

In TQDBItem
As above but additionally provides a new blank item.

In TQDBView
As above but also puts the panel into insertion mode.

KeyExists method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
function KeyExists(Key : TKey) : boolean;
```

Description
Indicates whether a particular Key is present in the index. Unlike the ExactMatch method KeyExists doesn't have the overhead of moving ItemIndex or triggering any events.

Kill method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Kill;
```

Description
The Kill method is used to close and delete all traces of a QDB file. **Use with care!**

Kill generates a BeforeKill event before doing anything drastic to give you chance to change your mind. If a BeforeKill handler is not defined Kill will not function.

If a Kill is successful the QDB file is deleted and a OnKilled event triggered.

If the FileName property is not set (*i.e.*, Ready is false) the Kill method raises an exception.

LastItem method

Applies To
TQDB, TQDBItem, TQDBView

Declaration

```
procedure LastItem;
```

Description

In TQDB
The LastItem method moves to the last key in the index that matches the Filter pattern and sets EoF to true.

A call to LastItem, when it actually moves to a different item, generates an OnNavigate event.

In TQDBItem
As above but also loads the last item and parses it into fields.

In TQDBView
As above but also displays the last item in the Panel.

NextItem method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure NextItem;
```

Description

In TQDB
The NextItem method moves to the next item whose key matches the Filter pattern.

A call to NextItem, when it actually moves to a different item, generates an OnNavigate event. EoF may also be set (see AggressiveUpdate). If ItemIndex already points to the unfiltered last entry NextItem generates the EQDBOutOfBounds exception or, if a handler for it has been assigned, the WarnOutOfBounds event.

In TQDBItem
As above but also loads the next item and parses it into fields.

In TQDBView
As above but also displays the next item in the Panel.

OrphanToRecover method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
function OrphanToRecover : boolean;
```

Description
The OrphanToRecover method checks the system's temporary directory for any of QDB's working files. Such files should only exist when a QDB component is currently at work on a file but, if a QDB component happens to fail before it is properly closed (*e.g.*, on rebooting the system), the orphaned working files will persist.

If OrphanToRecover finds a set of working files it tests to see if they are currently in use. If not they are counted as orphans of a past QDB file and, maybe, recoverable. Genuine orphans cause OrphanToRecover to return true (and otherwise false).

Note, however, that the two working files belonging to any QDB file may not be in sync since changes to the item file happen immediately but changes to the key file occur only as a result of the Save and SaveAs methods. Recover will only restore the state of the QDB file at the last save.

Pack method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
procedure Pack;
```

Description

The Pack method tidies up the QDB file on disk by rearranging items into key order and eliminating any that have been deleted.

Pack is also called implicitly by Compress and Expand, and when KeyCaseSensitive is changed from true to false.

PartialMatch method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
function PartialMatch(StartOfKey : TKey) : boolean;
```

Description
The PartialMatch method searches the list of keys for a match of the beginning of the key with the StartOfKey parameter. The search begins *after* the current item. If a match occurs ItemIndex is changed accordingly, the function returns true (otherwise false), and the OnFound event is triggered.

To catch a possible match with the first key in the list it is necessary to preface the first call to PartialMatch with PartialMatchInit.

See also: CloseMatch, ExactMatch, & PatternMatch.

PartialMatchInit method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure PartialMatchInit;
```

Description
A PartialMatch search always begins from the item after the current one. PartialMatchInit sets the ItemIndex to -1, the position before any items, so that PartialMatch can check the first key in the index.

N.B.: PartialMatchInit must always be followed immediately by a call to PartialMatch. Anything else is lilely to raise an EQDBOutOfBounds exception.

Pattern Matching

The Filter property and the PatternMatch method can operate in two modes depending upon the setting of UseGrepMatch. Both modes pay attention to KeyCaseSensitive but only grep-style matching is aware of the MatchWholeWord property.

Note that grep-style matching is *significantly* slower.

Simple pattern matching

The simple scheme uses with two wild cards, '<' and '>' (these wild cards can be changed using SetMatchChars):

'<' matches any run of characters at the start of a key

'>' matches any run of characters at the end of a key

Thus:

'hello there'	matches	'hello there'	but	'Hello There'	doesn't
'<there'	matches	'hello there'	but	'<the'	doesn't
'hello>'	matches	'hello there'	but	'ello>'	doesn't
'<lo th>'	matches	'hello there'	but	'<loth>'	doesn't

'<>' matches anything!

Grep-style pattern matching

The grep-style mode uses the following meta-characters:

'^'	matches only at the beginning of a key
'%'	matches only at the beginning of a word
'$'	matches only at the end of a key
'&'	matches only at the end of a word
'\'	quotes any character
'.'	matches any single character
':x'	matches any character of class x.
':a'	matches any alphabetic character
':d'	matches any numeric character
':n'	matches any alphanumeric character
': '	matches spaces, tabs, and other control characters
'[...]'	matches any character within the brackets
'[^...]'	matches any character but the ones which follow '^' within in the brackets

If you wish to employ a more elaborate matching scheme you can override the protected functions SimpleMatches and GrepMatches.

Robert R. Marsh, SJ & the British Province of the Society of Jesus

PatternMatch method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
function PatternMatch(Pattern : TKey) : boolean;
```

Description
The PatternMatch method searches the list of keys for a pattern match with the *pattern* parameter. The search begins after the current item. If a match occurs ItemIndex is changed accordingly, the function returns true (otherwise false), and the OnFound event is triggered.

To catch a possible match with the first key in the list precede the first call to PatternMatch with PatternMatchInit.

For other information on searching for keys: CloseMatch, ExactMatch, & PartialMatch.

PatternMatchInit method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure PatternMatchInit;
```

Description
A PatternMatch search always begins from the item after the current one. PatternMatchInit sets the ItemIndex to -1, the position before any items, so that PatternMatch can check the first key in the index.

N.B.: PatternMatchInit must always be followed immediately by a call to PatternMatch. Anything else is lilely to raise an EQDBOutOfBounds exception.

Post method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Post;
```

Description
Post belongs to the family of QDB methods corresponding to the buttons of TQDBNavigator. Post writes the new or modified item to the file. Post can be called directly or *via* TQDBNavigator.

Post behaves differently in the different descendants of TQDB as each descendant adds its own functionality.

In TQDB
Does nothing more than switch out of edit or insert mode and call UpdateNavigator.

In TQDBItem
As above but additionally stores the new or modified item to the file.

In TQDBView
As above but also puts the panel into the appropriate mode -- see AutoEdit.

PrepareToAdd method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure PrepareToAdd(NumberOfItems : TItemIndex);
```

Description
The PrepareToAdd method speeds up the process of adding a large batch of items. Call PrepareToAdd with NumberOfItems equal to the number of items you are about to add.

You do not need to use this method unless you are adding a large number of items at once or the QDB file is already very large (thousands of items).

The speed increase arises from allocating memory for the new items in one go rather than piecemeal.

PrevItem method

Applies To
TQDB, TQDBItem, TQDBView

Declaration

```
procedure PrevItem;
```

Description

In TQDB
The PrevItem method moves to the previous item in the list whose key matches the Filter pattern.

A call to PrevItem, when it actually moves to a different item, generates an OnNavigate event. BoF might also be set (see AggressiveUpdate). If ItemIndex already points to the unfiltered first entry PrevItem generates the EQDBOutOfBounds exception or, if a handler for it has been assigned, the WarnOutOfBounds event.

In TQDBItem
As above but also loads the previous item and parses it into fields.

In TQDBView
As above but also displays the previous item in the Panel.

Recover method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
procedure Recover(NewFileName : string);
```

Description

The Recover method checks for QDB files that were not closed properly by a previous QDB component (see OrphanToRecover). If Recover finds such orphaned files, and the current QDB component is not already working on a QDB file, the orphans will be reconstituted as a new QDB file called NewFileName. It is the programmers responsibility to ensure NewFileName is appropriate.

Note, however, that the recovered QDB file will only reflect the state of the file when last saved -- intermediate modifications are lost.

Refresh method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure Refresh;
```

Description
Refresh belongs to the family of QDB methods corresponding to the buttons of TQDBNavigator. Refresh reloads the current item abandoning any changes. Refresh can be called directly or *via* TQDBNavigator.

Refresh behaves differently in the different descendants of TQDB as each descendant adds its own functionality.

In TQDB
Does nothing at all.

In TQDBItem
Undoes any changes to the current item by reloading it from the file.

In TQDBView
As above.

Save method

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
procedure Save;
```

Description

The list of keys for a QDB file is stored in memory but new items themselves are written directly to the working QDB file. This means that the index portion of the QDB file (which is only read once at load-time) is usually out of sync with the corresponding data portion.

The Save method writes out the list of keys to the file bringing the two portions back into harmony. Then, if any disaster should befall your computer, the two working files will still be available and can be recovered (see OrphanToRecover and Recover).

If you want to trade data security for execution speed, Save may be called after every change to the database.

During the processing a ProgressUpdate event is generated at intervals.

If the FileName property is not set (*i.e.*, Ready is false) the Save method raises an EQDBNoFile exception.

SaveAs method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure SaveAs(NewName : string);
```

Description
The SaveAs method saves the current state of the QDB file to a new file given by NewName. It is the user's responsibility to ensure that NewName doesn't conflict with an existing file which would be overwritten.

During the processing a ProgressUpdate event is generated at intervals.

SetMatchChars method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure SetMatchChars(Front : char; back : char);
```

Description
The simple pattern-matching scheme counts certain characters in a pattern as "wild." By default, '<' matches any number of characters at the beginning of a key and '>' any number at the end.

SetMatchChars lets you chose different characters to play these roles. For example, a more "traditional" choice might be:

```
       SetMatchChars('^', '$');
```

SimpleMatches method

Applies to
TQDB

Declaration
```
function SimpleMatches(key: TKey; pattern: TKey): boolean;
```

Description
The SimpleMatches method implements the simple QDB pattern-matching protocol. It is a protected method that cannot be called directly by your programs but can be overridden if you wish to implement a more sophisticated scheme.

Be aware, however, that pattern-matching *via* the Filter property already incurs quite an overhead -- one you may not want to increase with a complex matching routine.

If you override SimpleMatches you should also override SetMatchChars.

UpdateNavigator method

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
procedure UpdateNavigator;
```

Description
If a QDBNavigator is controlling the navigation of a QDB component or one of its descendants some form of feedback is necessary to let the navigator update the state of its buttons to reflect the state of the QDB file, *e.g.*, to disable the edit button when the file is ReadOnly.

UpdateNavigator instructs the attached QDBNavigator (if there is one) to update itself. This method is called automatically by the "button" methods (*e.g.*, FirstItem or Post) but not, for performance reasons, when setting ItemIndex directly or using one of the search or match methods. You can, however, call UpdateNavigator yourself when you judge it to be appropriate.

BeforeKill event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property BeforeKill : TConfirmEvent;
```

Description
When the Kill method is executed the BeforeKill event is triggered before any drastic action is taken.

A handler for this event can present a confirmation request to the user to ensure the database isn't destroyed by accident. Such a handler passes a boolean true/false in the TConfirmEvent's OK parameter.

N.B.: For safety's sake if no handler for this event is assigned it will appear that Kill is not functioning.

BeforeOverWrite event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property BeforeOverWrite : TConfirmEvent;
```

Description
Once a QDB file is opened all actions take place on a *copy* of its contents. Actions that would commit the copied data back to the original QDB file need to ensure that the file has not been opened in the meantime by some other program or TQDB component.

Such actions (setting the FileName property, freeing the component, calling the SaveAs method, *etc.*) trigger a BeforeOverWrite event. A handler for this event can present a confirmation request to get the user's permission to go ahead with overwriting the original even though it is in use. Such a handler passes a boolean true/false in the TConfirmEvent's OK parameter.

If no handler is installed, or if OK is set to false, the file is not overwriten. Instead a new QDB file is created with a new name derived from the original by the following algorithm:

> "add as many '1.' prefixes to the file name as necessary to produce a unique filename"

e.g., QDBDemo.qdb would become 1.QDBDemo.qdb or, if this latter name were already in use, 1.1.QDBDemo.qdb, etc.

OnAdded event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property OnAdded : TNotifyEvent;
```

Description
When a new item has been added successfully to the database the OnAdded event occurs.

A handler for this event could be used, for example, to update a screen.

OnChanged event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property OnChanged : TNotifyEvent;
```

Description
When an existing item has been successfully changed the OnChanged event occurs.

OnDeleted event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property OnDeleted : TNotifyEvent;
```

Description
When an item has been deleted from the database the OnDeleted event occurs. Delete removes a key from the list of keys but does not remove the item stored in the physical QDB file. Items are only physically removed when the Pack method is called.

OnDemandPassword event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property OnDemandPassword : TPasswordEvent;
```

Description
On setting FileName to point to an encrypted QDB file a PassWord is required. If the PassWord property is set correctly the file is opened normally but if the PassWord blank the OnDemandPassword event is triggered to give the user the oppotunity to enter an appropriate value.

OnFileAssigned event

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
property OnFileAssigned : TNotifyEvent;
```

Description

When the FileName property has been successfully changed the OnFileAssigned event occurs.

OnFound event

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
property OnFound : TNotifyEvent;
```

Description

The four search methods, CloseMatch, ExactMatch, PartialMatch and PatternMatch, each attempt to locate a specific key in the list of keys. If they are successful the OnFound event is triggered.

OnKilled event

Applies To

TQDB, TQDBItem, TQDBView

Declaration

```
property OnKilled : TNotifyEvent;
```

Description

When a QDB file has been successfully deleted by the Kill method the OnKilled event occurs.

OnNavigate event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property OnNavigate : TNotifyEvent;
```

Description
Whenever the index pointer is successfully moved the OnNavigate event is triggered. Changing the ItemIndex property directly or *via* FirstItem, LastItem, NextItem or PrevItem triggers OnNavigate, as do Add and Delete and the searching methods CloseMatch, ExactMatch, PartialMatch, and PatternMatch. The Items and ItemsByKey array properties may also trigger this event.

If, however, the above methods are sandwiched in a BeginUpdate - EndUpdate block OnNavigate is not triggered. This can speed up repeated operations by cutting out the screen updating that usually happens in the OnNavigate event.

N.B.: This event is only triggered if the ItemIndex property actually changes. The BoF and EoF properties, which are set when ItemIndex does not change, should never be tested in OnNavigate event handlers.

ProgressUpdate event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property ProgressUpdate : TProgressEvent;
```

Description
Actions on the database that might be lengthy generate the ProgressUpdate event at intervals during the action. The TProgressEvent message includes the percentage completion of the action and the kind of action that is going on.

WarnNoData event

Applies To
TQDB, TQDBItem, TQDBView

Declaration

```
property WarnNoData : TWarningEvent;
```

Description
Any action that attempts to adjust the ItemIndex of an ***empty*** QDB file triggers an EQDBNoData exception or, if a handler has been assigned for it, the WarnNoData event. You can choose to respond in either way.

If you wish to ignore all such events you must supply an empty WarnNoData event handler.

WarnOutOfBounds event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property WarnOutOfBounds : TWarningEvent;
```

Description
Attempts to navigate the ItemIndex before the first item or after the last item generate an EQDBOutOfBounds exception or, if a handler has been assigned for it, the WarnOutOfBounds event. You can choose to respond in either way.

If you wish to ignore all such events you must supply an empty WarnOutOfBounds event handler.

In an empty QDB file the WarnNoData event takes priority over WarnOutOfBounds.

WarnReadOnly event

Applies To
TQDB, TQDBItem, TQDBView

Declaration
```
property WarnReadOnly : TWarningEvent;
```

Description
Whenever an attempt is made to modify a ReadOnly QDB file it triggers an EQDBReadOnly exception or, if a handler has been assigned for it, the WarnReadOnly event. You can choose to respond in either way.

If you wish to ignore all such events you must supply an empty WarnReadOnly event handler.

The Add, Change, Delete, & Kill methods, for example, will not operate when the ReadOnly property is true.

TQDBNavigator Component

Properties Events Tasks

Unit
QDB

Description
TQDBNavigator was written in Borland Delphi by Robert R. Marsh, S.J.

This is Version 2.1.

Please let me know:

of any bugs you discover
what functionality you would like to see added
how you might like to see something done differently

I can be contacted at rrm@sprynet.com. Check out my web site: http://home.sprynet.com/sprynet/rrm/

Properties

▶ Run-time only

🔑 Key properties

In TQDBNavigator

	Flat	
▶		Glyphs
		Hints
	🔑	Orientation
	🔑	QDB
	🔑	VisibleButtons

Events

Key events

In TQDBNavigator

- BeforeAction
- OnCancel
- OnClick
- OnDelete
- OnEdit
- OnFirst
- OnInsert
- OnLast
- OnNext
- OnPost
- OnPrev
- OnRefresh

Flat property

Applies to
TQDBNavigator

Declaration
```
property Flat : boolean;
```

Description
Sets the navigator buttons to the flat, flyover style. Only available when the operating system supports it, *i.e.,* in Windows 95+ with suitable DLLs present.

Glyphs property

Applies to

TQDBNavigator

Declaration

```
property Glyphs[Btn : TNavigateBtn] : Graphics.TBitmap;
```

Description

Allows the default button bitmaps to be replaced with ones of the user's choice.

Run-time only.

Hints property

Applies to

TQDBNavigator

Declaration

```
property Hints : TStrings;
```

Description

Allows you to set the hints for navigator buttons. By default the hints correspond to the button names as stored in the qdb resource file.

Orientation property

Applies to

TQDBNavigator

Declaration

```
property Orientation : TNavOrientation;
```

Description

Governs orientation of the navigator:

noHoriz	horizontal
noVert	vertical
noAuto	if width > height then horizontal else vertical

QDB property

Applies to

TQDBNavigator

Declaration

```
property QDB : TQDB;
```

Description

The QDB property is used to connect a QDBNavigator control to a particular database which can be of type TQDB or any of its descendants.

VisibleButtons property

Applies to
TQDBNavigator

Declaration
```
property VisibleButtons : TButtonSet;
```

Description
The navigator buttons can be hidden by manipulating the VisibleButtons set. By default all the buttons are visible.

See also: TNavigateBtn.

BeforeAction event

Applies To

TQDBNavigator

Declaration

```
property BeforeAction : TNavClickEvent;
```

Description

When any of the navigator buttons are pressed three things happen:

first the BeforeAction event is triggered

next the default action or event occurs

finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnCancel event

Applies To
TQDBNavigator

Declaration
`property OnCancel : TBtnPressEvent;`

Description
When the cancel button (nbCancel) is pressed three things happen:

> first the BeforeAction event is triggered
>
> next if an OnCancel event handler has been assigned OnCancel is triggered and if not the attached QDB's Cancel method is called.
>
> finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnClick event

Applies To
TQDBNavigator

Declaration
```
property OnClick : TNavClickEvent;
```

Description
When any of the navigator buttons are pressed three things happen:

first the BeforeAction event is triggered

next the default action or event occurs

finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnDelete event

Applies To

TQDBNavigator

Declaration

```
property OnDelete : TBtnPressEvent;
```

Description

When the delete button (nbDelete) is pressed three things happen:

> first the BeforeAction event is triggered
>
> next if an OnDelete event handler has been assigned OnDelete is triggered and if not the attached QDB's Delete method is called.
>
> finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnEdit event

Applies To
TQDBNavigator

Declaration
```
property OnEdit : TBtnPressEvent;
```

Description
When the edit button (nbEdit) is pressed three things happen:

first the BeforeAction event is triggered

next if an OnEdit event handler has been assigned OnEdit is triggered and if not the attached QDB's Edit method is called.

finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnFirst event

Applies To
TQDBNavigator

Declaration
`property OnFirst : TBtnPressEvent;`

Description
When the first button (nbFirst) is pressed three things happen:

first the BeforeAction event is triggered

next if an OnFirst event handler has been assigned OnFirst is triggered and if not the attached QDB's FirstItem method is called.

finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnInsert event

Applies To
TQDBNavigator

Declaration
`property OnInsert : TBtnPressEvent;`

Description
When the insert button (nbInsert) is pressed three things happen:

> first the BeforeAction event is triggered
>
> next if an OnInsert event handler has been assigned OnInsert is triggered and if not the attached QDB's Insert method is called.
>
> finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnLast event

Applies To
TQDBNavigator

Declaration
```
property OnLast : TBtnPressEvent;
```

Description
When the last button (nbLast) is pressed three things happen:

first the BeforeAction event is triggered

next if an OnLast event handler has been assigned OnLast is triggered and if not the attached QDB's LastItem method is called.

finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnNext event

Applies To
TQDBNavigator

Declaration
`property OnNext : TBtnPressEvent;`

Description
When the next button (nbNext) is pressed three things happen:

> first the BeforeAction event is triggered
>
> next if an OnNext event handler has been assigned OnNext is triggered and if not the attached QDB's NextItem method is called.
>
> finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnPost event

Applies To
TQDBNavigator

Declaration
`property OnPost : TBtnPressEvent;`

Description
When the post button (nbPost) is pressed three things happen:

> first the BeforeAction event is triggered
>
> next if an OnPost event handler has been assigned OnPost is triggered and if not the attached QDB's Post method is called.
>
> finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnPrev event

Applies To
TQDBNavigator

Declaration
`property OnPrev : TBtnPressEvent;`

Description
When the prev button (nbPrev) is pressed three things happen:

> first the BeforeAction event is triggered
>
> next if an OnPrev event handler has been assigned OnPrev is triggered and if not the attached QDB's PrevItem method is called.
>
> finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

OnRefresh event

Applies To
TQDBNavigator

Declaration
`property OnRefresh : TBtnPressEvent;`

Description
When the refresh button (nbRefresh) is pressed three things happen:

> first the BeforeAction event is triggered
>
> next if an OnRefresh event handler has been assigned OnRefresh is triggered and if not the attached QDB's Refresh method is called.
>
> finally the OnClick event is triggered

This give some flexibility in responding to navigator buttons.

TQDBItem Component

Properties Methods Events Tasks

Unit
QDBView

Description
TQDBItem was written in Borland Delphi by Robert R. Marsh, S.J.

This is Version 2.1.

Please let me know:

> of any bugs you discover
> what functionality you would like to see added
> how you might like to see something done differently

I can be contacted at rrm@sprynet.com. Check out my web site: http://home.sprynet.com/sprynet/rrm/

Properties

▶ Run-time only
🔑 Key properties

In TQDBItem

▶		AsBoolean
▶		AsDateTime
▶		AsInteger
▶		AsReal
▶		AsString
▶		
🔑	AutoEdit	
		DateTimeFormatStr
▶		
🔑	FieldCount	
▶		
🔑	FieldNames	
▶		
🔑	FieldTypes	
		RealFormatStr

In TQDB

		AboutAuthor
		AboutVersion
▶		AdminAsBoolean
▶		AdminAsInteger
🔑		AdminAsString
		AggressiveUpdate
🔑		
🔑	BoF	
		CacheFrequency
		CacheSize
		Compression
🔑		
🔑	Count	
🔑		CurrentItem
🔑		
🔑	EoF	
	🔑	FileName
		Filter
🔑		FilteredCount
🔑		
🔑	ItemIndex	
🔑		Items
🔑		ItemsByKey
🔑		ItemSize
🔑		
🔑	Key	
🔑		KeyCaseSensitive
🔑		MatchWholeWord
🔑		Password
	🔑	ReadOnly
🔑		Ready
		SaveOnClose
		UseGrepMatch

Methods

🔑 Key methods

In TQDBItem

- AddField
- Clear
- ClearStructure
- Fetch
- FetchStructure
- FieldIndex
- GetField
- ListFileFieldNames
- Store
- StoreAs
- StoreStructure

In TQDB

- 🔑 Add
- AddItem
- AddStreamItem
- AdminClear
- AdminCount
- AdminDelete
- AdminKeyExists
- AdminKeys
- 🔑 AssignKeyList
- BeginUpdate
- CacheFlush
- CacheStatistics
- Cancel
- 🔑 Change
- ChangeItem
- ChangeStreamItem
- 🔑 CloseMatch
- Compress
- 🔑 Delete
- DeleteItem
- Edit
- EndUpdate
- 🔑 ExactMatch
- Expand
- 🔑 FirstItem
- 🔑 Get
- GetItem
- GetStreamItem
- Insert
- 🔑 KeyExists
- Kill
- 🔑 LastItem
- 🔑 NextItem
- OrphanToRecover
- Pack
- 🔑 PartialMatch
- PartialMatchInit
- 🔑 PatternMatch

PatternMatchInit
Post
PrepareToAdd
PrevItem
Recover
Refresh
Save
SaveAs
SetMatchChars
UpdateNavigator

Events

🔑 Key events

In TQDBItem

- 🔑 OnKey

In TQDB

- BeforeKill
- BeforeOverWrite
- OnAdded
- OnChanged
- OnDeleted
- OnDemandPassword
- OnFileAssigned
- OnFound
- OnKilled
- 🔑 OnNavigate
- ProgressUpdate
- WarnNoData
- WarnOutOfBounds
- WarnReadOnly

AsBoolean property

Applies to
TQDBItem, TQDBView

Declaration
```
property AsBoolean[Index : integer] : boolean;
```

Description
Retrieves one of the the current item's fields as a boolean value, performing conversions if necessary (and possible).

If an unreasonable conversion is attempted (e.g., trying to convert a ftgraphic field) an EQDBFieldError exception is raised. If a reasonable conversion fails due to a bad value (e.g., a badly formed string) the exception raised is EQDBConvertError.

AsDateTime property

Applies to
TQDBItem, TQDBView

Declaration
```
property AsDateTime[Index : integer] : TDateTime;
```

Description
Retrieves one of the the current item's fields as a TDateTime value, performing conversions if necessary (and possible).

If an unreasonable conversion is attempted (e.g., trying to convert a ftgraphic field) an EQDBFieldError exception is raised. If a reasonable conversion fails due to a bad value (e.g., a badly formed string) the exception raised is EQDBConvertError.

AsInteger property

Applies to
TQDBItem, TQDBView

Declaration
```
property AsInteger[Index : integer] : longint;
```

Description
Retrieves one of the the current item's fields as an integer value, performing conversions if necessary (and possible).

If an unreasonable conversion is attempted (e.g., trying to convert a ftgraphic field) an EQDBFieldError exception is raised. If a reasonable conversion fails due to a bad value (e.g., a badly formed string) the exception raised is EQDBConvertError.

AsReal property

Applies to
TQDBItem, TQDBView

Declaration
```
property AsReal[Index : integer] : extended;
```

Description
Retrieves one of the the current item's fields as an extended value, performing conversions if necessary (and possible).

If an unreasonable conversion is attempted (e.g., trying to convert a ftgraphic field) an EQDBFieldError exception is raised. If a reasonable conversion fails due to a bad value (e.g., a badly formed string) the exception raised is EQDBConvertError.

AsString property

Applies to
TQDBItem, TQDBView

Declaration
```
property AsString[Index : integer] : string;
```

Description
Retrieves one of the the current item's fields as a string value, performing conversions if necessary (and possible).

If an unreasonable conversion is attempted (e.g., trying to convert a ftgraphic field) an EQDBFieldError exception is raised.

AutoEdit property

Applies to
TQDBItem, TQDBView

Declaration
```
property AutoEdit;
```

Description
By default moving to a different item switches QDBItem and QDBView out of editing mode but if AutoEdit is true the current editing mode is preserved on moving to a different item.

DateTimeFormatStr property

Applies to
TQDBItem, TQDBView

Declaration
```
property DateTimeFormatStr : string;
```

Description
Governs the conversion of TDateTime values to their string representation (by SysUtils.DateTimeFormat). The default value, 'c', uses the system short date format follwoed by the system long time format.

FieldCount property

Applies to
TQDBItem, TQDBView

Declaration
```
property FieldCount : integer;
```

Description
The number of fields as currently defined.

The list of fields may have been built in two distinct ways: it may have been loaded from a QDB file *via* FetchStructure; or it may have been constructed at run-time using the AddField method.

FieldNames property

Applies to
TQDBItem, TQDBView

Declaration
```
property FieldNames[Index : integer] : string;
```

Description
The names of the currently defined fields.

The list of fields may have been built in two distinct ways: it may have been loaded from a QDB file via FetchStructure; or it may have been constructed at run-time using the AddField method.

FieldTypes property

Applies to
TQDBItem, TQDBView

Declaration
```
property FieldTypes[Index : integer] : TQDBFieldType;
```

Description
The types of the currently defined fields. The field type indicates the internal structure of the data in the field.

The list of fields may have been built in two distinct ways: it may have been loaded from a QDB file via FetchStructure; or it may have been constructed at run-time using the AddField method.

RealFormatStr property

Applies to
TQDBItem, TQDBView

Declaration
```
property RealFormatStr : string;
```

Description
Governs the conversion of extended number to their string representation (by SysUtils.FormatFloat). By default the format string is empty.

AddField method

Applies To

TQDBItem, TQDBView

Declaration

```
procedure AddField(FieldName : string; FieldType : TQDBFieldType);
```

Description

Adds a field description to the in-memory field list at run time. Note that this can put the field list out of sync with the structure stored in the file. See also: StoreStructure and FetchStructure.

ClearStructure method

Applies To

TQDBItem, TQDBView

Declaration

```
procedure ClearStructure;
```

Description

A low-level method for removing the field structure information stored in the current QDB file.

Fetch method

Applies To
TQDBItem, TQDBView

Declaration
```
procedure Fetch;
```

Description
The Clear, Fetch, and Store methods operate at a lower level than the Cancel, Delete, Edit, Insert, Post, and Refresh methods and should generally be avoided unless necessary.

Fetch loads the current item from the file in a way appropriate to the component.

In TQDBItem
Loads the current item and parses it into fields.

In TQDBView
As above but also displays the item in the associated Panel.

FetchStructure method

Applies To
TQDBItem, TQDBView

Declaration
```
procedure FetchStructure;
```

Description
A low-level method for retrieving the current field definitions from the associated QDB file.

Note that TQDBItem stores less information about its fields than TQDBView

See also: ListFileFieldNames

FieldIndex method

Applies To
TQDBItem, TQDBView

Declaration
```
function FieldIndex(const Name : string) : integer;
```

Description
Returns the index of a field in the field list given its name.

GetField method

Applies To
TQDBItem, TQDBView

Declaration
```
function GetField(Index : integer) : TMemoryStream;
```

Description
Provides direct access to the fields of the current item. Each field is a memory stream. You should take great care in handling this stream. You can read from it and write to it at will but you must not destroy or free it and you should reset the stream pointer to the start of the stream when you have finished with it.

ListFileFieldNames method

Applies To
TQDBItem, TQDBView

Declaration
```
procedure ListFileFieldNames(Names : TStrings);
```

Description
Fills Names with just the names of the file's fields as stored in the file. Note this could be different from the values in the FieldNames property.

Store method

Applies To
TQDBItem, TQDBView

Declaration
```
procedure Store;
```

Description
The Clear, Fetch, and Store methods operate at a lower level than the Cancel, Delete, Edit, Insert, Post, and Refresh methods and should generally be avoided unless necessary.

Store saves the current item to the file in a way appropriate to the component.

In TQDBItem
Combines the current item's fields and saves them. An OnKey event handler must have been assigned to supply a key with which to index the item.

In TQDBView
As above but handles the item displayed in the associated Panel.

StoreAs method

Applies To
TQDBItem, TQDBView

Declaration
```
procedure StoreAs(NewKey : TKey);
```

Description
Combines the data in the current items fields into an item and stores it with the NewKey. This method does the job of the Store method but without needing to trigger the OnKey event.

StoreStructure method

Applies To

TQDBItem, TQDBView

Declaration

```
procedure StoreStructure;
```

Description

A low-level method for storing the current field definitions in the associated QDB file.

Note that TQDBItem stores less information about its fields than TQDBView.

OnKey event

Applies To

TQDBItem, TQDBView

Declaration

`property OnKey : TKeyEvent;`

Description

The Store method calls the OnKey event to supply the key necessary to index the item. If no OnKey handler has been assigned the EQDBKeyError exception is raised.

Example

```
procedure OnKeyHandler(Sender : TObject; var Key : TKey);
begin
  Key := Q.AsString['KeyField'];
end;
```

TQDBView Component

Properties Methods Events Tasks

Unit
QDBView

Description
TQDBView was written in Borland Delphi by Robert R. Marsh, S.J.

This is Version 2.1.

Please let me know:

  of any bugs you discover
  what functionality you would like to see added
  how you might like to see something done differently

I can be contacted at rrm@sprynet.com. Check out my web site: http://home.sprynet.com/sprynet/rrm/

Properties

🔑 Run-time only
🔑 Key properties

In TQDBView

ActiveColor
ExcludeTag
InactiveColor
Panel

In TQDBItem

AsBoolean
AsDateTime
AsInteger
AsReal
AsString
AutoEdit
DateTimeFormatStr
FieldCount
FieldNames
FieldTypes
RealFormatStr

In TQDB

AboutAuthor
AboutVersion
AdminAsBoolean
AdminAsInteger
AdminAsString
AggressiveUpdate
BoF
CacheFrequency
CacheSize
Compression
Count
CurrentItem
EoF
FileName
Filter
FilteredCount
ItemIndex
Items
ItemsByKey
ItemSize
Key
KeyCaseSensitive

🔑		MatchWholeWord
🔑		Password
	🔑	ReadOnly
🔑		Ready
	SaveOnClose	
	UseGrepMatch	

Methods

🔑 Key methods

In TQDBView

- CheckStructure
- 🔑 RegisterControl
- RegisterGraphicFormat

In TQDBItem

- AddField
- Clear
- ClearStructure
- Fetch
- FetchStructure
- FieldIndex
- GetField
- ListFileFieldNames
- Store
- StoreAs
- StoreStructure

In TQDB

- 🔑 Add
- AddItem
- AddStreamItem
- AdminClear
- AdminCount
- AdminDelete
- AdminKeyExists
- AdminKeys
- 🔑 AssignKeyList
- BeginUpdate
- CacheFlush
- CacheStatistics
- Cancel
- 🔑 Change
- ChangeItem
- ChangeStreamItem
- 🔑 CloseMatch
- Compress
- 🔑 Delete
- DeleteItem
- Edit
- EndUpdate
- 🔑 ExactMatch
- Expand
- 🔑 FirstItem
- 🔑 Get
- GetItem
- GetStreamItem
- Insert
- 🔑 KeyExists
- Kill
- 🔑 LastItem
- 🔑 NextItem

OrphanToRecover
Pack
PartialMatch
PartialMatchInit
PatternMatch
PatternMatchInit
Post
PrepareToAdd
PrevItem
Recover
Refresh
Save
SaveAs
SetMatchChars
UpdateNavigator

Events

🔑 Key events

In TQDBItem

🔑 OnKey

In TQDB

- BeforeKill
- BeforeOverWrite
- OnAdded
- OnChanged
- OnDeleted
- OnDemandPassword
- OnFileAssigned
- OnFound
- OnKilled
- 🔑 OnNavigate
- ProgressUpdate
- WarnNoData
- WarnOutOfBounds
- WarnReadOnly

ActiveColor property

Applies to

TQDBView

Declaration

```
property ActiveColor : TColor;
```

Description

The color used to display a panel's contents when available for editing.

ExcludeTag property

Applies to
TQDBView

Declaration
```
property ExcludeTag : longint;
```

Description
Controls on a Panel are included among those to be stored unless they have their Tag property set to this value. By default ExcludeTag is -999 but it may be changed.

InactiveColor property

Applies to

TQDBView

Declaration

```
property InactiveColor : TColor;
```

Description

The color used to display a panel's contents when not available for editing.

Panel property

Applies to
TQDBView

Declaration
```
property Panel : TCustomPanel;
```

Description
The actual Panel which contains the components to be stored in the QDB file.

If a QDB file is open the panel is checked to see if its component structure matches that of the file. If there is a mismatch the EQDBPanelError exception is raised. If the open QDB file is empty it is branded with the structure defined by the panel.

CheckStructure method

Applies To

TQDBView

Declaration

```
function CheckStructure : boolean;
```

Description

Compares the file structure as stored in a QDB file to that defiend *via* the Panel property.

RegisterControl method

Applies To
TQDBView

Declaration
```
procedure RegisterControl(AClass : TControlClass; FieldType : TFieldType;
ClearProc, FetchProc, StoreProc : TClassProc);
```

Description
RegisterControl is used to tell TQDBView how to deal with a particular class of control (or its descendants). The TQDBView constructor uses RegisterControl to provide the default behavior for Edit boxes, listboxes, images, *etc*. You only need to bother with RegisterControl if you wish to override the default behavior or add a new type of control.

QDBView needs to be provided with procedures (of type TClassProc) to clear a control, to fetch its contents, and to store its contents.

For example, to register descendants of TCustomEdit use:

```
RegisterControl(TCustomEdit, ftstring, ClearCustomEdit, FetchCustomEdit,
StoreCustomEdit);
```

with:

```
// here Stream is irrelevant
procedure ClearCustomEdit(AControl : TControl; Stream : TStream);
begin
  // simply blank the control
  (AControl as TCustomEdit).Text := '';
end;

// Stream contains the data for this field only in whatever format you have
chosen
// Here data is stored as a simple string
procedure FetchCustomEdit(AControl : TControl; Stream : TStream);
var
  con : TCustomEdit;
  p : pchar;
  Len : longint;
begin
  try
    Len := Stream.size;
    if Len = 0 then
      Abort;
    con := (AControl as TCustomEdit);
    p := StrAlloc(Len);
    try
      Stream.ReadBuffer(p^, Len);
      con.SetTextBuf(p);
    finally
      StrDispose(p);
    end;
  except
    // if Fetch fails it should always Clear the control
    ClearCustomEdit(AControl, Stream);
  end;
```

```
end;

procedure StoreCustomEdit(AControl : TControl; Stream : TStream);
var
  con : TCustomEdit;
  Len : longint;
  p : pchar;
begin
  con := (AControl as TCustomEdit);
  Len := con.GetTextLen + 1;
  p := StrAlloc(Len);
  try
    con.GetTextBuf(p, Len);
    Stream.Write(p^, Len);
  finally
    StrDispose(p);
  end;
end;
```

RegisterGraphicFormat method

Applies To

TQDBView

Declaration

```
procedure RegisterGraphicFormat(const AExtension : string; AGraphicClass :
TGraphicClass);
```

Description

QDBView recognizes Delphi's built-in graphic formats (*e.g.*,bmp, wmf). If you extend Delphi's graphic unit to recognize other format, e.g., jpg you also have register them with QDBView using the RegisterGraphicFormat method.

EQDBBadKey type

Unit
QDB

Declaration
```
EQDBBadKey = class(EQDBError);
```

Description
Attempting to use a key which does not exist raises this exception.

EQDBConvertError type

Unit

QDBView

Declaration

```
EQDBConvertError = class(EQDBItemError)
```

Description

Exception raised when an implicit conversion fails in one of the AsXXXX properties.

EQDBFieldError type

Unit

QDBView

Declaration

```
EQDBFieldError = class(EQDBItemError)
```

Description

Exception raised when a field appears to be corrupt or if an unreasonable field type conversion is attempted.

EQDBFileError type

Unit
QDB

Declaration
```
EQDBFileError = class(EQDBError);
```

Description
Any error that arises with the physical QDB file raises this exception or one of its descendants..

EQDBIndexError type

Unit
QDB

Declaration
```
EQDBIndexError = class(EQDBError);
```

Description
Any error that arises with the QDB component's key list raises this exception or one of its descendants.

EQDBInvalidPW type

Unit
QDB

Declaration
```
EQDBInvalidPW = class(EQDBError);
```

Description
Attempting to open an encrypted QDB file without the correct Password raises this exception.

See also: OnDemandPassword.

EQDBItemError type

Unit

QDBView

Declaration

```
EQDBItemError = class(EQDBError)
```

Description

The general class of errors raised by TQDBItem. See also: EQDBConvertError, EQDBFieldError, EQDBKeyError.

EQDBKeyError type

Unit

QDBView

Declaration

```
EQDBKeyError = class(EQDBItemError)
```

Description

Exception raised if attempting to store an item without defining an OnKey event handler to provide a Key.

EQDBListError type

Unit
QDB

Declaration
```
EQDBListError = class(EQDBError);
```

Description
Any error that arises with QDB's internal lists raises this exception or one of its descendants.

EQDBNoCompress type

Unit
QDB

Declaration
```
EQDBNoCompress = class(EQDBError);
```

Description
Failure of the compression or expansion routines raises this exception.

EQDBNoData type

Unit
QDB

Declaration
```
EQDBNoData = class(EQDBIndexError);
```

Description
Attempting to operate on an empty file raises this exception or the corresponding warning event (if a handler for it has been assigned).

EQDBNoFile type

Unit
QDB

Declaration
```
EQDBNoFile = class(EQDBIndexError);
```

Description
If the QDB component is not Ready, *i.e.*, no file has been assigned, most operations raise this exception.

EQDBOutOfBounds type

Unit
QDB

Declaration
```
EQDBOutOfBounds = class(EQDBIndexError);
```

Description
Attempting to move ItemIndex outside the range 0..Count-1 raises this exception or the corresponding warning event (if a handler for it has been assigned).

EQDBViewError type

Unit

QDBView

Declaration

```
EQDBViewError = class(EQDBError)
```

Description

Exception raised when a TQDBView operation fails, in particular, when setting the Panel or FileName properties leads to a mismatch in file structure.

EQDBReadOnly type

Unit
QDB

Declaration
```
EQDBReadOnly = class(EQDBIndexError);
```

Description
Attempting to change a read-only file raises this exception or the corresponding warning event (if a handler for it has been assigned).

QDBTempFileLocation variable

Applies to
TQDB, TQDBItem, TQDBView

Declaration
```
var QDBTempFileLocation: string;
```

Description
QDB uses temporary files for many purposes, not least to hold the working copies of the QDB data and index files. By default these and other temporary files are stored in whatever directory Windows considers to be the proper location for temporary files. Generally this is the fastest drive available.

This can give rise to problems if, for example, the temporary directory is on a ram drive and very large temporary files have to be accomodated—an enormous amount of disk thrashing occurs as Windows tries to summon up enough virtual memory to do the job. In such cases it is better to force QDB's temporary files to be stored in another location.

If QDBTempFileLocation is '' (*i.e.*, an empty string—the default), or does not correspond to a valid directory, QDB will use the Windows default temporary directory. If QDBTempFileLocation corresponds to a valid directory QDB will store its files there.

N.B.: This does not work for Win16.

RenameOrMoveFile routine

Unit

QDB

Declaration

```
procedure RenameOrMoveFile(const SrcFileName, DstFileName : string);
```

Description

A file can be physically moved in two ways. "Copy and delete" is generally applicable but is much slower than "renaming" but the latter can only be used when the two filenames share a common logical drive. This routine manages the operation by the fastest means.

TempFileName routine

Unit
QDB

Declaration
```
function TempFileName(Prefix : string) : string;
```

Description
Given a three character prefix this routine generates a unique string suitable for use as a temporary filename.

TBtnPressEvent type

Unit

QDB

Declaration

```
TBtnPressEvent = procedure(Sender : TObject; Q : TQDB) of object;
```

Description

The handlers for the navigator's button-press events are of this type. If no event handler is assigned the buttons instead call the corresponding methods of the navigator's associated QDB.

TButtonSet type

Unit
QDB

Declaration
```
TButtonSet = set of TNavigateBtn;
```

Description
Used to determine which navigator buttons are visible.

TClassProc type

Unit

QDBView

Declaration

```
TClassProc = procedure(AControl : TControl; Stream : TStream);
```

Description

The type of procedure used by RegisterControl to teach TQDBView how to handle a class of control.

TConfirmEvent type

Unit
QDB

Declaration
```
TConfirmEvent = procedure(Sender : TObject; var OK : boolean) of object;
```

Description
Events of this type are used to elicit confirmation from the user that some operation should go ahead. See BeforeOverWrite and BeforeKill.

TDataIndex type

Unit
QDB

Declaration
```
TDataIndex = longint;
```

Description
Used to refer to the bytes of a data item.

TKey type

Unit
QDB

Declaration
```
TKey = string[255];
```

Description
Keys to the QDB index are Delphi short strings. Keys are held in ascending alphabetical order (with or without case-sensitivity). Each key must be unique.

TKeyEvent type

Unit

QDBView

Declaration

```
TKeyEvent = procedure(Sender : TObject; var Key : TKey) of object;
```

Description

The TKeyEvent is used to supply a key when an item needs to be stored. See also: OnKey

TItemIndex type

Unit
QDB

Declaration
```
TItemIndex = longint;
```

Description
Used to enumerate the items of the QDB file. The theoretical limit on the number of items is somewhat less than MaxLongInt (21,474,832) but you should be experiencing a shortage of memory before you get anywhere near this limit!

TNavClickEvent type

Unit
QDB

Declaration
```
TNavClickEvent = procedure(Sender : TObject; Button : TNavigateBtn) of object;
```

Description
The BeforeAction and OnClick events are of this type. Note that the Button which was pressed is passed as a parameter.

TNavGlyph type

Unit
QDB

Declaration
```
TNavGlyph = (ngEnabled, ngDisabled);
```

Description
Individual buttons, if visible, may be either enabled or disabled.

TNavigateBtn type

Unit
QDB

Declaration
```
TNavigateBtn = (nbFirst, nbPrev, nbNext, nbLast, nbInsert, nbDelete, nbEdit,
nbPost, nbCancel, nbRefresh);
```

Description
The buttons of QDBNavigator correspond to those of the Borland DBNavigator.

TNavOrientation type

Unit
QDB

Declaration
```
TNavOrientation = (noAuto, noHoriz, noVert);
```

Description
Governs the Orientation of display of the navigator

noHoriz	horizontal
noVert	vertical
noAuto	if width > height then horizontal else vertical

TPassword type

Unit
QDB

Declaration
```
TPassword = string[255];
```

Description
QDB passwords can be up to 255 characters long.

TPasswordEvent type

Unit
QDB

Declaration
```
TPasswordEvent = procedure(Sender : TObject; var Password : TPassword) of
object;
```

Description
Related to the OnDemandPassword event.

TPercentage type

Unit
QDB

Declaration
```
TPercentage = 0..100;
```

Description
The ProgressUpdate event (of type TProgressEvent) passes a parameter of type TPercentage to communicate the degree of completion of a (possibly) lengthy process.

TProgressEvent type

Unit

QDB

Declaration

```
TProgressEvent = procedure(Sender : TObject; Percent : TPercentage; Kind :
TProgressOrigin) of object;
```

Description

Lengthy processes trigger the ProgressUpdate event (of type TProgressEvent) at intervals to signal their degree of completion. Percent is the degree of completion and Kind refers to the process that is underway.

TProgressOrigin type

Unit
QDB

Declaration
```
TProgressOrigin = (prStart, prFinish, prSave, prPack, prKeyList, prCompress);
```

Description
The ProgressUpdate event passes a TProgressOrigin parameter to indicate which process is underway: prStart and prFinish indicate that a database file is being opened and closed respectively. The other constants refer to the methods and properties of similar name.

TQDBFileName type

Unit
QDB

Declaration
```
TQDBFileName = string;
```

Description
The type of a QDB FileName as used by its property editor.

TQDBFieldType type

Unit

QDBView

Declaration

```
TQDBFieldType = (ftunknown, ftinteger, ftintegers, ftreal,
    ftboolean, ftdatetime, ftstring, ftstrings, ftrichstrings,
    ftgraphic, ftthing);
```

Description

The field type indicates the internal structure of the field defined by a TQDBItem or TQDBView component.

ftinteger	4 byte longint value
ftintegers	a sequence of 4 byte integers terminated by longint(-1)
ftreal	10 byte extended value
ftboolean	4 byte boolean value
ftdatetime	8 byte TDateTime value (D2 style)
ftstring	null-terminated pchar
ftstrings	the output from TStrings.SaveToStream
ftrichstrings	the output from TRichEdit.SaveToStream
ftgraphic	3 byte signature then output from TGraphic.SaveToStream
ftthing	a generic blob

TWarningEvent type

Unit
QDB

Declaration
```
TWarningEvent = TNotifyEvent;
```

Description
Warning events are paired with corresponding exceptions. If no handler is assigned for these events an exception is raised instead of the warning event. If you want to avoid the exception you must assign a handler for the warning. In the simplest case this handler would do nothing.

The choice between these responses depends upon whether you want to centralize your code in an event handler or use try ... except blocks throughout your code. Both can be useful.

The warning events (with their corresponding exceptions) are:

WarnNoData	EQDBNoData
WarnOutOfBounds	EQDBOutOfBounds
WarnReadOnly	EQDBReadOnly