












## Getting started with InterBase Express

InterBase Express (IBX) is a set of data access components that provide a means of accessing data from InterBase databases. The following components are located on the InterBase tab of Delphi 5:

	<u>IBDatabase</u>
	<u>IBTransaction</u>
	<u>IBTable</u>
	<u>IBQuery</u>
	<u>IBDataSet</u>
	<u>IBStoredProc</u>
	<u>IBSQL</u>
	<u>IBUpdateSQL</u>
	<u>IBSQLMonitor</u>
	<u>IBDatabaseInfo</u>
	<u>IBEvents</u>

This document assumes that you are familiar with the Delphi development environment and know how to use the Standard, Data Access, and Data Control components.

Though they are similar to BDE components in name, the IBX components are somewhat different. For each component with a BDE counterpart, the sections below give a discussion of these differences.

There is no simple migration from BDE to IBX applications. Generally, you must replace BDE components with the comparable IBX components, and then recompile your applications. However, the speed you gain, along with the access you get to the powerful InterBase features make migration well worth your time.

### IBDatabase

Use an IBDatabase component to establish connections to databases, which can involve one or more concurrent transactions. Unlike BDE, IBX has a separate transaction component, which allows you to separate transactions and database connections.

To set up a database connection:

1. Drop an IBDatabase component onto a form or data module.
2. Fill out the DatabaseName property. For a local connection, this is the drive, path, and filename of the database file. Set the Connected property to True.
3. Enter a valid username and password and click OK to establish the database connection.

Tip: You can store the username and password in the IBDatabase component's Params property by setting the LoginPrompt property to False after logging in. For example, after logging in as the system administrator and setting the LoginPrompt property to False, you may see the following when editing the Params property:

```
user_name=sysdba
password=masterkey
```

### IBTransaction

Unlike the Borland Database Engine, IBX controls transactions with a separate component, IBTransaction. This powerful feature allows you to separate transactions and database connections, so you can take advantage of the InterBase two-phase commit functionality (transactions that span multiple connections) and multiple concurrent transactions using the same connection.

Use an IBTransaction component to handle transaction contexts, which might involve one or more database connections. In most cases, a simple one database/one transaction model will do.

To set up a transaction:

1. Set up an IBDatabase connection as described [above](#).
2. Drop an IBTransaction component onto the form or data module
3. Set the DefaultDatabase property to the name of your IBDatabase component.
4. Set the Active property to True to start the transaction.

### IBX dataset components

There are a variety of dataset components from which to choose with IBX, each having their own characteristics and task suitability:

#### IBTable

Use an [IBTable](#) component to set up a live dataset on a table or view without having to enter any SQL statements.

IBTable components are easy to configure:

1. Add an IBTable component to your form or data module.
2. Specify the associated database and transaction components.
3. Specify the name of the relation from the TableName drop-down list.
4. Set the Active property to True.

#### IBQuery

Use an [IBQuery](#) component to execute any InterBase DSQL statement, restrict your result set to only particular columns and rows, use aggregate functions, and join multiple tables.

IBQuery components provide a read-only dataset, and adapt well to the InterBase client/server environment. To set up an IBQuery component:

1. Set up an IBDatabase connection as described [above](#).
2. Set up an IBTransaction connection as described [above](#).
3. Add an IBQuery component to your form or data module.
4. Specify the associated database and transaction components.
5. Enter a valid SQL statement for the IBQuery's SQL property in the String list editor.
6. Set the Active property to True

#### IBDataSet

Use an [IBDataSet](#) component to execute any InterBase DSQL statement, restrict your result set to only particular columns and rows, use aggregate functions, and join multiple tables. IBDataSet components are similar to IBQuery components, except that they support live datasets without the need of an IBUpdateSQL component.

The following example that provides a live dataset for the COUNTRY table in employee.gdb:

1. Set up an IBDatabase connection as described [above](#).
2. Specify the associated database and transaction components.
3. Add an IBDataSet component to your form or data module.
4. Enter SQL statements for the following properties:

SelectSQL	SELECT Country, Currency FROM Country
RefreshSQL	SELECT Country, Currency FROM Country WHERE Country = :Country
ModifySQL	UPDATE Country SET Country = :Country, Currency = :Currency WHERE Country = :Old_Country
DeleteSQL	DELETE FROM Country WHERE Country = :Old_Country
InsertSQL	INSERT INTO Country (Country, Currency) VALUES (:Country, :Currency)

5. Set the Active property to True.

### **IBStoredProc**

Use IBStoredProc for InterBase executable procedures: procedures that return, at most, one row of information. For stored procedures that return more than one row of data, or "Select" procedures, use either IBQuery or IBDataSet components.

### **IBSQL**

Use an IBSQL component for data operations that need to be fast and lightweight. Operations such as data definition and pumping data from one database to another are suitable for IBSQL components.

In the following example, an IBSQL component is used to return the next value from a generator:

1. Set up an IBDatabase connection as described above.
2. Put an IBSQL component on the form or data module and set its Database property to the name of the database.
3. Add an SQL statement to the SQL property string list editor, for example:

```
SELECT GEN_ID(MyGenerator, 1) FROM RDB$DATABASE
```

### **IBUpdateSQL**

Use an IBUpdateSQL component to update read-only datasets. You can update IBQuery output with an IBUpdateSQL component:

1. Set up an IBQuery component as described above.
2. Add an IBUpdateSQL component to your form or data module.
3. Enter SQL statements for the following properties: DeleteSQL, InsertSQL, ModifySQL, and RefreshSQL.
4. Set the IBQuery component's UpdateObject property to the name of the IBUpdateSQL component.
5. Set the IBQuery component's Active property to True.

### **IBSQLMonitor**

Use an IBSQLMonitor component to develop diagnostic tools to monitor the communications between your application and the InterBase server. When the TraceFlags properties of an IBDatabase component are turned on, active IBSQLMonitor components can keep track of the connection's activity and send the output to a file or control.

A good example would be to create a separate application that has an IBSQLMonitor component and a Memo control. Run this secondary application, and on the primary application, activate the TraceFlags of the IBDatabase component. Interact with the primary application, and watch the second's memo control fill with data.

### **IBDatabaseInfo**

Use an IBDatabaseInfo component to retrieve information about a particular database, such as the sweep interval, ODS version, and the user names of those currently attached to this database.

For example, to set up an IBDatabaseInfo component that displays the users currently connected to the database:

1. Set up an IBDatabase connection as described above.
2. Put an IBDatabaseInfo component on the form or data module and set its Database property to the name of the database.
3. Put a Memo component on the form.
4. Put a Timer component on the form and set its interval.
5. Double click on the Timer's OnTimer event field and enter code similar to the following:

```
Memol.Text := IBDatabaseInfo.UserNames.Text;
```

### **IBEvents**

Use an IBEvents component to register interest in, and asynchronously handle, events posted by an InterBase server.

To set up an IBEvents component:

1. Set up an IBDatabase connection as described above.
2. Put an IBEvents component on the form or data module and set its Database property to the name of the database.
3. Enter events in the Events property string list editor, for example:

```
IBEvents.Events.Add('EVENT_NAME')
```

4. Set the Registered property to True.

## TIBBase

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBBase is the ancestor object from which TIBDatabase and TIBTransaction descend.

### Unit

IBDatabase

### Description

TIBBase is the ancestor object from which TIBDatabase and TIBTransaction descend.

## TIBBase properties

[TIBBase](#)

[Alphabetically](#) [Legend](#)

### In TIBBase

[Database](#)

▶ [DBHandle](#)



[Owner](#)

[Transaction](#)



[TRHandle](#)

TIBBase properties

[TIBBase](#)      [By object](#)      [Legend](#)

<u>Database</u>	
▶ <u>DBHandle</u>	
▶	<u>Owner</u>
	<u>Transaction</u>
▶	<u>TRHandle</u>

## TIBBase.Database

[TIBBase](#)

[See also](#)

Sets or returns the associated database.

**property** Database: TIBDatabase;

### Description

Use Database to set or return the associated database.



## TIBBase.DBHandle

[TIBBase](#)

[See also](#)

Indicates the database handle.

**property** DBHandle: PISC\_DB\_HANDLE;

### Description

Use DBHandle to return the database handle.

## TIBBase.Owner

[TIBBase](#)      [See also](#)

Indicates which component owns the component.

**property** Owner: TObject;

### Description

Owner refers to the SQL object (DataSet, TIBSQL, or Blob) that created the TIBBase component.

## TIBBase.Transaction

[TIBBase](#)

[See also](#)

Sets or returns the associated transaction.

**property** Transaction: TIBTransaction;

### Description

Use Transaction to set or return the associated transaction.

## TIBBase.TRHandle

[TIBBase](#)

[See also](#)

Returns the transaction handle.

**property** TRHandle: PISC\_TR\_HANDLE;

### Description

Use TRHandle to return the transaction handle.

## TIBBase events

[TIBBase](#)

[Alphabetically](#) [Legend](#)

### In TIBBase

- [OnAfterDatabaseDisconnect](#)
- ▶ [OnBeforeDatabaseDisconnect](#)
- ▶ [OnDatabaseFree](#)
- ▶ [OnAfterTransactionEnd](#)
- ▶ [OnBeforeTransactionEnd](#)
- ▶ [OnTransactionFree](#)

**TIBBase events**

<u>TIBBase</u>	<u>By object</u>	<u>Legend</u>
▶	<u>OnAfterDatabaseDisconnect</u>	
▶	<u>OnBeforeDatabaseDisconnect</u>	
▶	<u>OnDatabaseFree</u>	
▶	<u>OnAfterTransactionEnd</u>	
▶	<u>OnBeforeTransactionEnd</u>	
▶	<u>OnTransactionFree</u>	

## TIBBase.OnAfterDatabaseDisconnect

[TIBBase](#)      [See also](#)

Occurs after a database is disconnected.

**property** OnAfterDatabaseDisconnect: TNotifyEvent;

### Description

Write an OnAfterDatabaseDisconnect event handler to take specific actions after a database is disconnected.

## TIBBase.OnBeforeDatabaseDisconnect

[TIBBase](#)

[See also](#)

Occurs before a database disconnects.

**property** OnBeforeDatabaseDisconnect: TNotifyEvent;

### Description

Write an OnBeforeDatabaseDisconnect event handler to take specific actions before a database disconnects.



## TIBBase.DatabaseFree

[TIBBase](#)

[See also](#)

Occurs after a database is freed from memory.

**property** OnDatabaseFree: TNotifyEvent;

### Description

Write an OnDatabaseFree event handler to take specific actions after a database is freed from memory.

## TIBBase.OnAfterTransactionEnd

[TIBBase](#)      [See also](#)

Occurs after a transaction has ended.

**property** OnAfterTransactionEnd: TNotifyEvent;

### Description

Write an OnAfterTransactionEnd event handler to take specific actions after a transaction has ended.

## TIBBase.OnBeforeTransactionEnd

[TIBBase](#)    [See also](#)

Occurs before a transaction ends.

**property** OnBeforeTransactionEnd: TNotifyEvent;

### Description

Write an OnBeforeTransactionEnd event handler to take specific actions before a transaction ends.

## TIBBase.OnTransactionFree

[TIBBase](#)

[See also](#)

Occurs after a transaction has been freed from memory.

**property** OnTransactionFree: TNotifyEvent;

### Description

Write an OnTransactionFree event handler to take specific actions after a transaction has been freed from memory.

## **TIBBase methods**

[TIBBase](#)      [Alphabetically](#)

### **In TIBBase**

[CheckDatabase](#)  
[CheckTransaction](#)  
[Create](#)  
[Destroy](#)

### **Derived from TObject**

[AfterConstruction](#)  
[BeforeDestruction](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[DefaultHandler](#)  
[Dispatch](#)  
[FieldAddress](#)  
[Free](#)  
[FreeInstance](#)  
[GetInterface](#)  
[GetInterfaceEntry](#)  
[GetInterfaceTable](#)  
[InheritsFrom](#)  
[InitInstance](#)  
[InstanceSize](#)  
[MethodAddress](#)  
[MethodName](#)  
[NewInstance](#)

## TIBBase methods

[TIBBase](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[CheckDatabase](#)

[CheckTransaction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

## **TIBBase.CheckDatabase**

[TIBBase](#)      [See also](#)

Checks if the database is active.

**procedure** CheckDatabase;

### **Description**

Call CheckDatabase to check if the database is assigned and active.

## TIBBase.CheckTransaction

[TIBBase](#)

[See also](#)

Checks if the transaction is active.

**procedure** CheckTransaction;

### Description

Call CheckTransaction to check if the transaction is assigned and active.



## TIBBase.Create

[TIBBase](#)      [See also](#)

Creates an instance of an IBBase object.

**constructor** `Create (AOwner: TObject);`

### Description

Call Create to create an instance of an IBBase object.

## **TIBBase.Destroy**

[TIBBase](#)

[See also](#)

Destroys an IBBase object.

**destructor** Destroy;

### **Description**

Call Destroy to destroy an IBBase object and to free up any resources associated with it.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

**Hierarchy**

TObject

## TIBBatch

[Hierarchy](#)   [Properties](#)   [MethodsSee also](#)

TIBBatch is the abstract base class for the TIBBatchInput and TIBBatchOutput components.

### Unit

IBSQL

### Description

Use a TIBBatch object to provide properties and methods for use with the TIBBatchInput and TIBBatchOutput components, which make it possible to input and output data in virtually any format.

Descendents of this class can specify a file name (for input or output), and a TIBXSQLDA component representing a record or parameters. The ReadyFile method is called right before performing the batch input or output.

TIBBatch properties

[TIBBatch](#)      [Alphabetically Legend](#)

In TIBBatch

▶ [Columns](#)



[FileName](#)

[Params](#)

TIBBatch properties

[TIBBatch](#)   [By object](#)   [Legend](#)

▶ Columns

▶ FileName  
Params



## TIBBatch.Columns

[TIBBatch](#)      [See also](#)

Returns the XSQLDA columns.

**property** Columns: TIBXSQLDA;

### Description

Use the Columns property to retrieve the XSQLDA columns.

## TIBBatch.FileName

[TIBBatch](#)      [See also](#)

Sets or displays the name of the external file.

**property** FileName: String;

### Description

Use the FileName property to set or display the external file name.

## TIBBatch.Params

[TIBBatch](#)      [See also](#)

Returns the XSQLDA parameters.

**property** Params: TIBXSQLDA;

### Description

Use the Params property to retrieve the XSQLDA parameters.

## TIBBatch methods

[TIBBatch](#)

[Alphabetically](#)

### In TIBBatch

[Move](#)

[ReadyFile](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBBatch methods

[TIBBatch](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[Move](#)

[NewInstance](#)

[ReadyFile](#)

[SafeCallException](#)

## TIBBatch.Move

[TIBBatch](#)   [See also](#)

Indicates whether the can be moved.

**function** Move: Boolean;

### Description

Call Move to allow a file to be moved.

## TIBBatch.ReadyFile

[TIBBatch](#)    [See also](#)

Prepares the file.

**procedure** ReadyFile;

### Description

Call ReadyFile to prepare the file for input or output.

## Scope



Published

## Accessibility



Read-only



**Hierarchy**

TObject

## **TIBBatchInput**

[Hierarchy](#)   [Properties](#)   [Methods](#)[See also](#)

TIBBatchInput is the abstract class for performing all batch input.

### **Unit**

IBSQL

### **Description**

Use a TIBBatchInput object to provide properties and methods for performing all batch input.

**TIBBatchInput properties**

[TIBBatchInput](#) [Alphabetically](#) [Legend](#)

Derived from TIBBatch

▸ <u>Columns</u>	<u>FileName</u>
▸	<u>Params</u>

TIBBatchInput properties

[TIBBatchInput](#) [By object](#) [Legend](#)

▸ Columns

▸ FileName  
Params

## TIBBatchInput methods

[TIBBatchOutput](#)

[Alphabetically](#)

### In TIBBatchInput

[ReadParameters](#)

### Derived from TIBBatch

[Move](#)

[ReadyFile](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBBatchInput methods

[TIBBatch](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[Move](#)

[NewInstance](#)

[ReadParameters](#)

[ReadyFile](#)

[SafeCallException](#)

## TIBBatchInput.ReadParameters

[TIBBatchInput](#) [See also](#)

Reads the XSQLDA input parameters.

**function** ReadParameters: Boolean;

### Description

Call ReadParameters to read the input parameters of the extended SQL descriptor area (XSQLDA) from the specified file.

## Scope



Published

## Accessibility



Read-only



## Hierarchy

TObject

|

TIBBatch

## **TIBBatchOutput**

[Hierarchy](#)   [Properties](#)   [Methods](#)[See also](#)

TIBBatchOutput is the abstract class for performing all batch output.

### **Unit**

IBSQL

### **Description**

Use a TIBBatchOutput object to provide properties and methods for performing all batch output.

**TIBBatchOutput properties**

[TIBBatchOutput](#)

[Alphabetically Legend](#)

**Derived from TIBBatch**

▶ [Columns](#)



[FileName](#)

[Params](#)

TIBBatchOutput properties

[TIBBatchOutput](#)

[By object](#)

[Legend](#)

▸ [Columns](#)



[FileName](#)

[Params](#)

## TIBBatchOutput methods

[TIBBatchOutput](#)

[Alphabetically](#)

### In TIBBatchOutput

[WriteColumns](#)

### Derived from TIBBatch

[Move](#)

[ReadyFile](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBBatchOutput methods

[TIBBatch](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[Move](#)

[NewInstance](#)

[ReadyFile](#)

[SafeCallException](#)

[WriteColumns](#)

## TIBBatchOutput.WriteColumns

[TIBBatchOutput](#)

[See also](#)

Outputs the data in columns in the XSQLDA to the specified file.

**function** WriteColumns: Boolean

### Description

Call WriteColumns to output data in columns in the extended SQL descriptor area (XSQLDA) to the specified file.

## Scope



Published

## Accessibility



Read-only



## Hierarchy

TObject



TIBBatch

## TIBBCDField

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBBCDField encapsulates the Windows Currency type.

### Unit

IBCustomDataSet

### Description

TIBBCDField encapsulates the Windows Currency type.

## TIBBCDField properties

[TIBBCDField](#) [Alphabetically](#) [Legend](#)

### In TIBBCDField

► [Size](#)

### Derived from TBCDField

[AsCurrency](#)

[AsFloat](#)

[AsInteger](#)

[AsString](#)

[AsVariant](#)

► [Currency](#)

► [DataSize](#)

► [MaxValue](#)

► [MinValue](#)

► [Precision](#)

[Value](#)

### Derived from TNumericField

► [DisplayFormat](#)

► [EditFormat](#)

### Derived from TField

► [Alignment](#)

[AsBoolean](#)

[AsDateTime](#)

[AttributeSet](#)

► [AutoGenerateValue](#)

[Calculated](#)

► [CanModify](#)

► [ConstraintErrorMessage](#)

► [CurValue](#)

► [CustomConstraint](#)

[DataSet](#)

► [DataType](#)

► [DefaultExpression](#)

► [DisplayLabel](#)

► [DisplayName](#)

► [DisplayText](#)

► [DisplayWidth](#)

[EditMask](#)

► [EditMaskPtr](#)

► [FieldKind](#)

► [FieldName](#)

► [FieldNo](#)

► [FullName](#)

► [HasConstraints](#)

► [ImportedConstraint](#)

► [Index](#)

► [IsIndexField](#)

► [IsNull](#)

► [KeyFields](#)

[Lookup](#)

► [LookupCache](#)

- ▶ [LookupDataSet](#)
- ▶ [LookupKeyFields](#)
- ▶ ▶ [LookupList](#)
- ▶ [LookupResultField](#)
- ▶ [NewValue](#)
- ▶ ▶ [Offset](#)
- ▶ ▶ [OldValue](#)
- ▶ [Origin](#)
- ▶ [ParentField](#)
- ▶ [ProviderFlags](#)
- ▶ [ReadOnly](#)
- ▶ [Required](#)
- ▶ [Text](#)
- ▶ [ValidChars](#)
- ▶ [Visible](#)

#### **Derived from TComponent**

- ▶ [ComObject](#)
  - ▶ [ComponentCount](#)
  - ▶ [ComponentIndex](#)
  - ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
  - ▶ [DesignInfo](#)
- ▶ [Name](#)
- ▶ ▶ [Owner](#)
- ▶ [Tag](#)
- ▶ [VCLComObject](#)

## TIBBCDField properties

[TIBBCDField](#)

[By object](#)

[Legend](#)

▶	<a href="#">Alignment</a>
	<a href="#">AsBoolean</a>
	<a href="#">AsCurrency</a>
	<a href="#">AsDateTime</a>
	<a href="#">AsFloat</a>
	<a href="#">AsInteger</a>
	<a href="#">AsString</a>
	<a href="#">AsVariant</a>
	<a href="#">AttributeSet</a>
▶	<a href="#">AutoGenerateValue</a>
	<a href="#">Calculated</a>
▶	<a href="#">CanModify</a>
▶	<a href="#">ComObject</a>
▶	<a href="#">ComponentCount</a>
	<a href="#">ComponentIndex</a>
▶	<a href="#">Components</a>
▶	<a href="#">ComponentState</a>
▶	<a href="#">ComponentStyle</a>
▶	<a href="#">ConstraintErrorMessage</a>
▶	<a href="#">Currency</a>
▶	<a href="#">CurValue</a>
▶	<a href="#">CustomConstraint</a>
	<a href="#">DataSet</a>
▶	<a href="#">DataSize</a>
▶	<a href="#">DataType</a>
▶	<a href="#">DefaultExpression</a>
	<a href="#">DesignInfo</a>
▶	<a href="#">DisplayFormat</a>
▶	<a href="#">DisplayLabel</a>
▶	<a href="#">DisplayName</a>
▶	<a href="#">DisplayText</a>
▶	<a href="#">DisplayWidth</a>
▶	<a href="#">EditFormat</a>
	<a href="#">EditMask</a>
▶	<a href="#">EditMaskPtr</a>
▶	<a href="#">FieldKind</a>
▶	<a href="#">FieldName</a>
▶	<a href="#">FieldNo</a>
▶	<a href="#">FullName</a>
▶	<a href="#">HasConstraints</a>
▶	<a href="#">ImportedConstraint</a>
▶	<a href="#">Index</a>
▶	<a href="#">IsIndexField</a>
▶	<a href="#">IsNull</a>
▶	<a href="#">KeyFields</a>
	<a href="#">Lookup</a>
▶	<a href="#">LookupCache</a>
▶	<a href="#">LookupDataSet</a>
▶	<a href="#">LookupKeyFields</a>
▶	<a href="#">LookupList</a>
▶	<a href="#">LookupResultField</a>

- ▶ MaxValue
- ▶ MinValue
- ▶ Name
- ▶ NewValue
- ▶ Offset
- ▶ OldValue
- ▶ Origin
- ▶ Owner
- ▶ ParentField
- ▶ Precision
- ▶ ProviderFlags
- ▶ ReadOnly
- ▶ Required
- ▶ Size
- ▶ Tag
- ▶ Text
- ▶ ValidChars
- ▶ Value
- ▶ VCLComObject
- ▶ Visible

## **TIBBCDField.Size**

[TIBBCDField](#) [See also](#)

Indicates the length of the datatype.

**property** Size;

### **Description**

Indicates the length of the datatype. The default length is 8.

## TIBBCDField events

[TIBBCDField](#) [Alphabetically Legend](#)

### Derived from TField

- ▶ [OnChange](#)
- ▶ [OnGetText](#)
- ▶ [OnSetText](#)
- ▶ [OnValidate](#)



**TIBBCDField events**

[TIBBCDField](#)   [By object](#)   [Legend](#)

▢	<u>OnChange</u>
▢	<u>OnGetText</u>
▢	<u>OnSetText</u>
▢	<u>OnValidate</u>

## **TIBBCDField methods**

[TIBBCDField](#) [Alphabetically](#)

### **In TIBBCDField**

[Create](#)

### **Derived from TField**

[Assign](#)

[AssignValue](#)

[Clear](#)

[Destroy](#)

[FocusControl](#)

[GetData](#)

[IsBlob](#)

[IsValidChar](#)

[RefreshLookupList](#)

[SetData](#)

[SetFieldType](#)

[Validate](#)

### **Derived from TComponent**

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### **Derived from TPersistent**

[Assign](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

Free

FreeInstance

GetInterface

GetInterfaceEntry

GetInterfaceTable

InheritsFrom

InitInstance

InstanceSize

MethodAddress

MethodName

NewInstance

## TIBBCDField methods

[TIBBCDField](#) [By object](#)

[AfterConstruction](#)

[Assign](#)

[AssignValue](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Clear](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecuteAction](#)

[FieldAddress](#)

[FindComponent](#)

[FocusControl](#)

[Free](#)

[FreeInstance](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetData](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetParentComponent](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)

[InstanceSize](#)

[IsBlob](#)

[IsValidChar](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[RefreshLookupList](#)

[RemoveComponent](#)

SafeCallException

SetData

SetFieldType

UpdateAction

Validate

## TIBBCDField.Create

[TIBBCDField](#) [See also](#)

Creates an instance of a TIBBCDField object.

**constructor** `Create (AOwner: TComponent);`

### Description

Create sets the datatype to ftBCD and the size to 8.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published



## Hierarchy

TObject



TPersistent



TComponent



TField



TStringField

## TIBBlobStream

[Hierarchy](#)   [Properties](#)   [Methods](#)   [See also](#)

TIBBlobStream is a stream object that lets applications read from or write to field objects that represent Blob fields.

### Unit

IBBlob

### Description

Use TIBBlobStream to access or modify the value of a Blob field object. Blob field objects are TBlobField objects and descendants of TBlobField such as TGraphicField and TMemoField. Blob fields use Blob streams to implement many of their data access properties and methods.

TIBBlobStream allows objects that have no specialized knowledge of how data is stored in a Blob field to read or write such data by employing the uniform stream interface.

To use a Blob stream, create an instance of TIBBlobStream, use the methods of the stream to read or write the data, and then free the Blob stream. Do not use the same instance of TIBBlobStream to access data from more than one record. Instead, create a new TIBBlobStream object every time you need to read or write Blob data on a new record.

## TIBBlobStream properties

[TIBBlobStreamAlphabetically](#) [Legend](#)

### In TIBBlobStream

BlobID

▶ BlobMaxSegmentSize

▶ BlobNumSegments

▶ BlobSize

▶ BlobType

▶ Database

▶ DBHandle

▶ Handle

▶ Mode

▶ Modified

▶ Transaction

▶ TRHandle

### Derived from TStream

Position

Size

## TIBBlobStream properties

[TIBBlobStreamBy object](#)

[Legend](#)

BlobID

▶ BlobMaxSegmentSize

- ▶ BlobNumSegments
- ▶ BlobSize
- ▶ BlobType
- ▶ Database
- ▶ DBHandle
- ▶ Handle
- ▶ Mode
- ▶ Modified
- ▶ Position
- ▶ Size
- ▶ Transaction
- ▶ TRHandle

## TIBBlobStream.BlobID

[TIBBlobStream](#) [See also](#)

Sets or returns the Blob ID.

**property** BlobID: TISC\_QUAD;

### Description

Use BlobID to set or return the 64-bit system-defined Blob ID, which is stored in a field in the table and points to the first segment of the Blob or to a page of pointers to Blob fragments

## TIBBlobStream.BlobMaxSegmentSize

[TIBBlobStream](#) [See also](#)

Returns the maximum segment size.

**property** BlobMaxSegmentSize: Long;

### Description

Use BlobMaxSegmentSize to return the length of the longest Blob segment.

## TIBBlobStream.BlobNumSegments

[TIBBlobStream](#) [See also](#)

Returns the total number of segments in the Blob.

**property** BlobNumSegments: Long;

### Description

Use BlobNumSegments to return the total number of segments in the Blob.

## TIBBlobStream.BlobSize

[TIBBlobStream](#) [See also](#)

Returns the total size of the Blob.

**property** BlobSize: Long;

### Description

Use BlobSize to return the total size of the Blob in bytes.



## TIBBlobStream.BlobType

[TIBBlobStream](#) [See also](#)

Returns the Blob type.

**property** BlobType: Short;

### Description

Use BlobType to return the Blob type; either 0 for segmented, or 1 for stream.

## TIBBlobStream.Database

[TIBBlobStream](#) [See also](#)

Sets or returns the associated database.

**property** Database: TIBDatabase;

### Description

Use Database to set or return the associated database.

## TIBBlobStream.DBHandle

[TIBBlobStream](#) [See also](#)

Indicates the database handle.

**property** DBHandle: PISC\_DB\_HANDLE;

### Description

Use DBHandle to return the database handle.

## TIBBlobStream.Handle

[TIBBlobStream](#) [See also](#)

Indicates the Blob handle.

**property** Handle: TISC\_BLOB\_HANDLE;

### Description

Use Handle to return the Blob handle.

## TIBBlobStream.Mode

[TIBBlobStream](#) See also

Sets or returns the Blob stream mode type.

**type** TBlobStreamMode = **set of** (bmRead, bmWrite, bmReadWrite);

**property** Mode: TBlobStreamMode;

### Description

Use Mode to set or return the Blob stream mode type. BlobStreamMode can be one of the following values.

Value	Meaning
bmRead	The stream is used to read from a Blob field
bmWrite	The stream is used to write to a Blob field
bmReadWrite	The stream is used to read from or write to a Blob field

## TIBBlobStream.Modified

[TIBBlobStream](#) See also

Indicates whether or not the Blob field has been modified.

**property** Modified: Boolean;

### Description

Modified returns True when the value to a Blob field has been changed. If the value of the Blob field is set by using the properties of TBlobField, or by using a TIBBlobStream object, Modified is automatically set to True.

## TIBBlobStream.Transaction

[TIBBlobStream](#) [See also](#)

Sets or returns the associated transaction.

**property** Transaction: TIBTransaction;

### Description

Use Transaction to set or return the associated transaction.

## TIBBlobStream.TRHandle

[TIBBlobStream](#) [See also](#)

Returns the transaction handle.

**property** TRHandle: PISC\_TR\_HANDLE;

### Description

Use TRHandle to return the transaction handle.

## TIBBlobStream methods

### [TIBBlobStreamAlphabetically](#)

#### In TIBBlobStream

[Call](#)  
[CheckReadable](#)  
[CheckWritable](#)  
[Create](#)  
[Destroy](#)  
[Finalize](#)  
[LoadFromFile](#)  
[LoadFromStream](#)  
[Read](#)  
[SaveToFile](#)  
[SaveToStream](#)  
[Seek](#)  
[SetSize](#)  
[Truncate](#)  
[Write](#)

#### Derived from TStream

[CopyFrom](#)  
[ReadBuffer](#)  
[ReadComponent](#)  
[ReadComponentRes](#)  
[ReadResHeader](#)  
[WriteBuffer](#)  
[WriteComponent](#)  
[WriteComponentRes](#)  
[WriteDescendent](#)  
[WriteDescendentRes](#)

#### Derived from TObject

[AfterConstruction](#)  
[BeforeDestruction](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[DefaultHandler](#)  
[Dispatch](#)  
[FieldAddress](#)  
[Free](#)  
[FreeInstance](#)  
[GetInterface](#)



GetInterfaceEntry

GetInterfaceTable

InheritsFrom

InitInstance

InstanceSize

MethodAddress

MethodName

NewInstance

## TIBBlobStream methods

### TIBBlobStreamBy object

AfterConstruction  
BeforeDestruction  
Call  
CheckReadable  
CheckWritable  
ClassInfo  
ClassName  
ClassNamels  
ClassParent  
ClassType  
CleanupInstance  
CopyFrom  
Create  
DefaultHandler  
Destroy  
Dispatch  
FieldAddress  
Finalize  
Free  
FreeInstance  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
InheritsFrom  
InitInstance  
InstanceSize  
LoadFromFile  
LoadFromStream  
MethodAddress  
MethodName  
NewInstance  
Read  
ReadBuffer  
ReadComponent  
ReadComponentRes  
ReadResHeader  
SaveToFile  
SaveToStream  
Seek  
SetSize  
Truncate  
Write

WriteBuffer

WriteComponent

WriteComponentRes

WriteDescendent

WriteDescendentRes

## TIBBlobStream.Call

[TIBBlobStreamSee also](#)

Returns an error message based on the error code.

**function** Call(ISC\_STATUS; RaiseError: Boolean): ISC\_STATUS;

### Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning an error based on the value of RaiseError.

## TIBBlobStream.CheckReadable

[TIBBlobStream](#) [See also](#)

Indicates whether or not a Blob is readable.

**procedure** CheckReadable;

### Description

Call CheckReadable to determine whether or not a Blob is readable. This method raises an exception if the Blob is not readable.

## **TIBBlobStream.CheckWritable**

[TIBBlobStream](#) [See also](#)

Indicates whether or not a Blob is write-able.

**procedure** CheckWritable;

### **Description**

Call CheckWritable to determine whether or not a Blob is write-able. This method raises an exception if the Blob is not write-able.

## TIBBlobStream.Create

[TIBBlobStream](#) [See also](#) [Example](#)

Creates an instance of TIBBlobStream.

**constructor** Create;

### Description

Call Create to obtain an instance of TIBBlobStream for reading from or writing to a specific TBlobField object.

## TIBBlobStream.Destroy

[TIBBlobStream](#) [See also](#)

Destroys an instance of TIBBlobStream.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead call Free. Free verifies that the TIBBlobStream object is not already freed and only then calls Destroy.

Destroy the TIBBlobStream object by calling Free when it is no longer needed for reading from or writing to the BLOB field.

Destroy triggers an OnDataChange event if the Blob stream was used to overwrite or modify the data in the field. Destroy then frees any buffers that were allocated to handle the data.



## **TIBBlobStream.Finalize**

### [TIBBlobStream](#)

Creates a Blob on the InterBase server and writes the data from the BlobStream to it.

**procedure** Finalize;

### **Description**

Finalize creates a Blob on the InterBase server and writes the data from the BlobStream to the Blob and then closes the Blob.

## TIBBlobStream.LoadFromFile

[TIBBlobStream](#) [See also](#)

Loads a Blob from a file to the field.

**procedure** LoadFromFile (Filename: **String**);

### Description

Use LoadFromFile to load the contents of a file into a Blob field. Specify the name of the file to load into the field as the value of the FileName parameter.

## TIBBlobStream.LoadFromStream

[TIBBlobStream](#) [See also](#)

Loads a Blob from a stream into the field.

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

### Description

Use LoadFromStream to copy the contents of a stream into the Blob field. Specify the stream from which the field's value is copied as the value of the Stream parameter.

## TIBBlobStream.Read

[TIBBlobStreamSee also](#)

Reads up to Count bytes from the current position in the field's data into Buffer.

**function** Read(**var** Buffer; Count: Longint): Longint;

### Description

Call Read to read data from the Blob field when the number of bytes in the field's data is not known. Buffer must have at least Count bytes allocated to hold the data that was read from the field.

Read transfers up to Count bytes from the Blob data into Buffer, starting in the current position, and then advances the current position by the number of bytes actually transferred. Read returns the number of bytes actually transferred (which may be less than the number requested in Count.)

Read checks the Transliterate property of the field, and converts the data into ANSI from the character set specified by the dataset if Transliterate is True.

All the other data-reading methods of a Blob stream (ReadBuffer, ReadComponent) call Read to do their actual reading.

**Note:** Do not call Read when the TIBBlobStream was created in bmWrite mode.

## TIBBlobStream.SaveToFile

[TIBBlobStream](#) [See also](#)

Saves the contents of the Blob field to a file.

**procedure** SaveToFile (FileName: **string**);

### Description

Use SaveToFile to save the contents of the Blob field to a file. Specify the name of the file as the value of the FileName parameter.

## TIBBlobStream.SaveToStream

[TIBBlobStream](#) [See also](#)

Saves the contents of the BLOB field to a stream.

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

### Description

Use SaveToStream to copy the contents of a Blob field to a stream. Specify the name of the stream to which the field's value is saved as the value of the Stream parameter.

## TIBBlobStream.Seek

[TIBBlobStream](#) [See also](#)

Resets the current position of the TIBBlobStream object.

**function** Seek(Offset: Longint; Origin: Word): Longint;

### Description

Use Seek to move the current position within the Blob data by the indicated offset. Seek allows an application to read from or write to a particular location within the Blob data.

The Origin parameter indicates how to interpret the Offset parameter. Origin should be one of the following values:

Value	Meaning
soFromBeginning	<ul style="list-style-type: none"><li>• Offset is from the beginning of the Blob data</li><li>• Seek moves to the position Offset</li><li>• Offset must be <math>\geq 0</math></li></ul>
soFromCurrent	<ul style="list-style-type: none"><li>• Offset is from the current position in the Blob data</li><li>• Seek moves to Position + Offset</li></ul>
soFromEnd	<ul style="list-style-type: none"><li>• Offset is from the end of the Blob data</li><li>• Offset must be <math>\leq 0</math> to indicate a number of bytes before the end of the Blob</li></ul>

Seek returns the new value of the Position property, the new current position in the Blob data.

## **TIBBlobStream.SetSize**

[TIBBlobStream](#) [See also](#)

Set the new total size of the Blob.

**procedure** SetSize (NewSize: Long) ;

### **Description**

Call SetSize to set the new total size of the Blob.



## TIBBlobStream.Truncate

[TIBBlobStream](#) [See also](#)

Discards all data in the Blob field from the current position on.

**procedure** Truncate;

### Description

Use Truncate to limit the size of the Blob data. Calling Truncate when the current position is 0 will clear the contents of the Blob field.

**Note:** Do not call Truncate when the TIBBlobStream was created in bmRead mode.

## TIBBlobStream.Write

[TIBBlobStream](#) [See also](#)

Writes Count bytes from Buffer to the current position in the field and updates the current position by Count bytes.

```
function Write(const Buffer; Count: Longint): Longint;
```

### Description

Use Write to write Count bytes to the Blob field, starting at the current position. The Write method for TIBBlobStream always writes the entire Count bytes, as Blob data does not necessarily include a termination character. Thus, Write is equivalent to the WriteBuffer method.

Write checks the Transliterate property of the field, and converts the data from ANSI into the character set of the dataset if Transliterate is True.

All the other data-writing methods of a Blob stream (WriteBuffer, WriteComponent) call Write to do their actual writing.

**Note:** Do not call Write when the TIBBlobStream was created in bmRead mode.

## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



TStream

## **TIBCustomDataSet**

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

Encapsulates InterBase Express functionality for descendent dataset objects.

### **Unit**

IBCustomDataSet

### **Description**

TIBCustomDataSet is a dataset object that defines InterBase Express (IBX) functionality for a dataset. Applications never use TIBCustomDataSet objects directly. Instead they use the descendants of TIBCustomDataSet, such as TIBDataSet, TIBQuery, TIBStoredProc, and TIBTable, which inherit its dataset-related properties and methods.

## TIBCustomDataSet properties

[TIBCustomDataSet](#)

[Alphabetically Legend](#)

### In TIBCustomDataSet

- ▶ [CachedUpdates](#)
- ▶ [Database](#)
  - ▶ [DBHandle](#)
  - ▶ [Transaction](#)
  - ▶ [TRHandle](#)
- ▶ [UpdateObject](#)
- ▶ [UpdateRecordTypes](#)
- ▶ [UpdatesPending](#)

### Derived from TDataSet

- ▶ [Active](#)
  - ▶ [AggFields](#)
- ▶ [AutoCalcFields](#)
  - ▶ [Bof](#)
  - ▶ [Bookmark](#)
  - ▶ [DatasetField](#)
  - ▶ [DataSource](#)
  - ▶ [DefaultFields](#)
- ▶ [Designer](#)
  - ▶ [Eof](#)
  - ▶ [FieldCount](#)
  - ▶ [FieldDefList](#)
  - ▶ [FieldDefs](#)
  - ▶ [FieldList](#)
  - ▶ [Fields](#)
  - ▶ [FieldValues](#)
- ▶ [Found](#)
  - ▶ [Modified](#)
- ▶ [Name](#)
- ▶ [ObjectView](#)
  - ▶ [SparseArrays](#)
  - ▶ [State](#)

### Derived from TComponent

- ▶ [ComObject](#)
  - ▶ [ComponentCount](#)
  - ▶ [ComponentIndex](#)
  - ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
  - ▶ [DesignInfo](#)
- ▶ [Owner](#)
- ▶ [Tag](#)
- ▶ [VCLComObject](#)

## TIBCustomDataSet properties

[TIBCustomDataSet](#)

[By object](#)

[Legend](#)

- ▶ [Active](#)
- ▶ ▶ [AggFields](#)
- ▶ ▶ [AutoCalcFields](#)
- ▶ ▶ [Bof](#)
- ▶ ▶ [Bookmark](#)
- ▶ ▶ [CachedUpdates](#)
- ▶ ▶ [ComObject](#)
- ▶ ▶ [ComponentCount](#)
- ▶ [ComponentIndex](#)
- ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
- ▶ [Database](#)
  - ▶ [DataSource](#)
  - ▶ [DBHandle](#)
  - ▶ [DefaultFields](#)
- ▶ [Designer](#)
  - ▶ [DesignInfo](#)
  - ▶ [Eof](#)
  - ▶ [FieldCount](#)
  - ▶ [FieldDefList](#)
  - ▶ [FieldDefs](#)
  - ▶ [FieldList](#)
  - ▶ [Fields](#)
- ▶ [FieldValues](#)
- ▶ [Found](#)
  - ▶ [Modified](#)
- ▶ [Name](#)
- ▶ [ObjectView](#)
- ▶ ▶ [Owner](#)
- ▶ ▶ [SparseArrays](#)
- ▶ ▶ [State](#)
- ▶ ▶ [Tag](#)
- ▶ ▶ [TRHandle](#)
- ▶ [Transaction](#)
- ▶ [UpdateObject](#)
- ▶ [UpdateRecordTypes](#)
- ▶ ▶ [UpdatesPending](#)
- ▶ [VCLComObject](#)

## TIBCustomDataSet.CachedUpdates

[TIBCustomDataSet](#)

[See also](#)

Specifies whether cached updates are enabled for a dataset.

**property** CachedUpdates: Boolean;

### Description

CachedUpdates enables or disables the use of cached updates for a dataset. If CachedUpdates is True, cached updates are enabled. If CachedUpdates is False, cached updates are disabled.

When cached updates are enabled, updates to a dataset (such as posting changes, inserting new records, or deleting records), are stored in an internal cache on the client machine instead of being written directly to the dataset's underlying database tables. When changes are complete, an application writes all cached changes to the database in the context of a single transaction.

Cached updates are most useful to client applications in two-tiered applications. The main benefits of enabling cached updates are:

- Fewer transactions and shorter transaction times.
- Minimization of network traffic.

The potential drawbacks of enabling cached updates are:

- Other applications can access and change the actual data on the server while users are editing local copies of the data, resulting in an update conflict when cached updates are applied to the database.
- Other applications cannot access data changes made by an application until its cached updates are applied to the database.

**Note:** Instead of using cached updates, applications can obtain the same benefits with greater control by using a client dataset.



## TIBCustomDataSet.Database

[TIBCustomDataSet](#)

[See also](#)

Identifies the database component for which this dataset represents one or more tables.

**property** Database: TIBDatabase;

### Description

Use Database to access the properties, events, and methods of the database component associated with this dataset.

## **TIBCustomDataSet.DBHandle**

[TIBCustomDataSet](#)

[See also](#)

Specifies the database handle for the dataset.

**property** DBHandle: PISC\_DB\_HANDLE;

### **Description**

Check DBHandle to determine the database handle for the dataset.

## **TIBCustomDataSet.Transaction**

[TIBCustomDataSet](#)

[See also](#)

Identifies the transaction under which the query executes.

**property** Transaction: [TIBTransaction](#);

### **Description**

Use Transaction to determine under which transaction the query executes.

## **TIBCustomDataSet.TRHandle**

[TIBCustomDataSet](#)

[See also](#)

Specifies the transaction handle for the dataset.

**property** TRHandle: PISC\_TR\_HANDLE;

### **Description**

Check TRHandle to determine the transaction handle for the dataset.

## TIBCustomDataSet.UpdateObject

[TIBCustomDataSet](#)

[See also](#)

Specifies the update object component used to update a read-only result set when cached updates are enabled.

**property** UpdateObject: TIBDataSetUpdateObject;

### Description

Use UpdateObject to specify the TIBDataSetUpdateObject component to use in an application that must be able to update a read-only result set.

In some cases, such as a query made against multiple tables, a live result set cannot be returned. In these cases, UpdateObject can be used to specify a TIBUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

## TIBCustomDataSet.UpdateRecordTypes

[TIBCustomDataSet](#)

[See also](#)

[Example](#)

Specifies the type of records visible in a dataset when cached updates are enabled.

```
type TIBUpdateRecordTypes = set of (cusModified, cusInserted, cusDeleted,  
    cusUnmodified, cusUninserted);
```

```
property UpdateRecordTypes: TIBUpdateRecordTypes;
```

### Description

Use UpdateRecordTypes to specify the records that are visible in a dataset when cached updates are enabled. UpdateRecordTypes is a set that can have the following values:

Value	Meaning
cusModified	Modified records are visible.
cusInserted	Inserted records are visible.
cusDeleted	Deleted records are visible.
cusUnmodified	Unmodified records are visible.
cusUninserted	Uninserted records are visible.

By default, a dataset is created with an UpdateRecordTypes set of cusModified, cusInserted, or cusUnmodified, meaning that all existing, edited, or inserted records are visible to the user.

An application that must cycle through a dataset to undelete records may change UpdateRecordTypes as part of an undelete method, so that deleted records are “visible” long enough to restore them to their previously undeleted conditions.

Similarly, an application that must cycle through a dataset to uninsert records may change UpdateRecordTypes as part of an uninsert method, so that uninserted records are “visible” long enough to restore them to their previously inserted conditions.

An application might also use UpdateRecordTypes like a filter to temporarily limit visible records to those added or inserted by the user during the current session.

## TIBCustomDataSet.UpdatesPending

[TIBCustomDataSet](#)

[See also](#)

Indicates whether the cached updates buffer contains records that are not yet applied.

**property** UpdatesPending: Boolean;

### Description

Examine UpdatesPending to check the status of the cached updates buffer. If UpdatesPending is True, then there are edited, deleted, or inserted records to apply to the database. If UpdatesPending is False, there are no records in the cache.

## TIBCustomDataSet events

[TIBCustomDataSet](#)

[Alphabetically Legend](#)

### In TIBCustomDataSet

- ▶ [OnUpdateError](#)
- ▶ [OnUpdateRecord](#)

### Derived from TDataSet

- ▶ [AfterCancel](#)
- ▶ [AfterClose](#)
- ▶ [AfterDelete](#)
- ▶ [AfterEdit](#)
- ▶ [AfterInsert](#)
- ▶ [AfterOpen](#)
- ▶ [AfterPost](#)
- ▶ [AfterRefresh](#)
- ▶ [AfterScroll](#)
- ▶ [BeforeCancel](#)
- ▶ [BeforeClose](#)
- ▶ [BeforeDelete](#)
- ▶ [BeforeEdit](#)
- ▶ [BeforeInsert](#)
- ▶ [BeforeOpen](#)
- ▶ [BeforePost](#)
- ▶ [BeforeRefresh](#)
- ▶ [BeforeScroll](#)
- ▶ [OnCalcFields](#)
- ▶ [OnDeleteError](#)
- ▶ [OnEditError](#)
- ▶ [OnNewRecord](#)
- ▶ [OnPostError](#)



## TIBCustomDataSet events

TIBCustomDataSet

By object

Legend

▶	<u>AfterCancel</u>
▶	<u>AfterClose</u>
▶	<u>AfterDelete</u>
▶	<u>AfterEdit</u>
▶	<u>AfterInsert</u>
▶	<u>AfterOpen</u>
▶	<u>AfterPost</u>
▶	<u>AfterRefresh</u>
▶	<u>AfterScroll</u>
▶	<u>BeforeCancel</u>
▶	<u>BeforeClose</u>
▶	<u>BeforeDelete</u>
▶	<u>BeforeEdit</u>
▶	<u>BeforeInsert</u>
▶	<u>BeforeOpen</u>
▶	<u>BeforePost</u>
▶	<u>BeforeRefresh</u>
▶	<u>BeforeScroll</u>
▶	<u>OnCalcFields</u>
▶	<u>OnDeleteError</u>
▶	<u>OnEditError</u>
▶	<u>OnNewRecord</u>
▶	<u>OnPostError</u>
▶	<u>OnUpdateError</u>
▶	<u>OnUpdateRecord</u>

## TIBCustomDataSet.OnUpdateError

[TIBCustomDataSet](#)

[See also](#)

Occurs if an exception is generated when cached updates are applied to a database.

### type

```
TIBUpdateAction = (uaFail, uaAbort, uaSkip, uaRetry, uaApplied, uaApply);  
TIBUpdateErrorEvent = procedure (DataSet: TDataSet; E: EDatabaseError;  
UpdateKind: TUpdateKind; var UpdateAction: TIBUpdateAction) of object;
```

**property** OnUpdateError: TIBUpdateErrorEvent;

### Description

Write an OnUpdateError event handler to respond to exceptions generated when cached updates are applied to a database.

Because there is a delay between the time a record is first cached and the time cached updates are applied, there is a possibility that another application may change one or more of the same records in the database before the cached changes can be applied. DataSet is the name of the dataset to which updates are applied.

E is a pointer to a EDBEngineError object from which an application can extract an error message and the actual cause of the error condition. An OnUpdateError handler can use this information to determine how to respond to the error condition.

UpdateKind indicates whether the error occurred while inserting, deleting, or modifying a record.

UpdateAction indicates the action to take when the OnUpdateError handler exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateError can handle or correct the error, set UpdateAction to uaRetry before exiting the error handler. The following table lists the possible values for UpdateAction and what they indicate:

Value	Meaning
uaAbort	Aborts the update operation without Returning an error message
uaApply	For internal use only
uaApplied	Not used in error handling routines
uaFail	Aborts the update operation and returns an error message
uaRetry	Repeats the update operation that originally raised the error condition
uaSkip	Skips updating the record that raised the error condition, and leaves the unapplied changes in the cache

The error handler can use the TField.OldValue and TField.NewValue properties to evaluate error conditions and set TField.NewValue to a new value to reapply. In this case, set UpdateAction to uaRetry before exiting.

**Note:** If a call to ApplyUpdates raises an exception and ApplyUpdates is not called within the context of a try..except block, an error message is Returned. If an OnUpdateError handler cannot correct the error condition and leaves UpdateAction set to uaFail, the error message is Returned twice. To prevent reReturn, set UpdateAction to uaAbort in the error handler.

**Important:** The code in an OnUpdateError handler must not call any methods that make a different record the current one.

## TIBCustomDataSet.OnUpdateRecord

[TIBCustomDataSet](#)

[See also](#)

Occurs when cached updates are applied to a record.

### type

```
TIBUpdateAction = (uaFail, uaAbort, uaSkip, uaRetry, uaApply, uaApplied);  
TIBUpdateRecordEvent = procedure (DataSet: TIBDataSet; UpdateKind:  
TUpdateKind; var UpdateAction: TIBUpdateAction) of object;
```

**property** OnUpdateRecord: TIBUpdateRecordEvent;

### Description

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

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

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

UpdateAction indicates the action taken by the OnUpdateRecord handler before it exits. On entry into the handler, UpdateAction is always set to uaFail. If OnUpdateRecord is successful, it should set UpdateAction to uaApplied before exiting. The following table lists the possible values for UpdateAction and what they indicate:

Value	Meaning
uaAbort	Abort the update operation without Returning an error message.
uaApply	For internal use.
uaApplied	Update is applied. Free update record from cache.
uaFail	Aborts the update operation and Returns an error message.
uaRetry	Not used for record updates.
uaSkip	Update is skipped. Leave update record in the cache.

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

## **TIBCustomDataSet methods**

[TIBCustomDataSet](#)

[Alphabetically](#)

### **In TIBCustomDataSet**

[ApplyUpdates](#)

[BatchInput](#)

[BatchOutput](#)

[CachedUpdateStatus](#)

[CancelUpdates](#)

[Create](#)

[CreateBlobStream](#)

[Destroy](#)

[FetchAll](#)

[GetCurrentRecord](#)

[GetFieldData](#)

[Locate](#)

[LocateNext](#)

[Lookup](#)

[RecordModified](#)

[RevertRecord](#)

[Undelete](#)

[UpdateStatus](#)

### **Derived from TDataSet**

[ActiveBuffer](#)

[Append](#)

[AppendRecord](#)

[CheckBrowseMode](#)

[ClearFields](#)

[Close](#)

[CompareBookmarks](#)

[ControlsDisabled](#)

[CursorPosChanged](#)

[Delete](#)

[DisableControls](#)

[Edit](#)

[EnableControls](#)

[FieldByName](#)

[FindField](#)

[FindFirst](#)

[FindLast](#)

[FindNext](#)

[FindPrior](#)

[First](#)

[FreeBookmark](#)

[GetBookmark](#)

GetDetailDataSets  
GetDetailLinkFields  
GetFieldList  
GetFieldNames  
GetProviderAttributes  
GotoBookmark  
Insert  
InsertRecord  
IsEmpty  
IsLinkedTo  
Last  
MoveBy  
Next  
Open  
Post  
Prior  
Refresh  
Resync  
SetFields  
Translate  
UpdateCursorPos  
UpdateRecord

#### **Derived from TComponent**

DestroyComponents  
Destroying  
ExecuteAction  
FindComponent  
FreeNotification  
FreeOnRelease  
GetNamePath  
GetParentComponent  
HasParent  
InsertComponent  
RemoveComponent  
SafeCallException  
UpdateAction

#### **Derived from TPersistent**

Assign

#### **Derived from TObject**

AfterConstruction  
BeforeDestruction  
ClassInfo  
ClassName  
ClassNames

ClassParent  
ClassType  
CleanupInstance  
DefaultHandler  
Dispatch  
FieldAddress  
Free  
FreeInstance  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
InheritsFrom  
InitInstance  
InstanceSize  
MethodAddress  
MethodName  
NewInstance

## TIBCustomDataSet methods

[TIBCustomDataSet](#)

[By object](#)

[ActiveBuffer](#)  
[AfterConstruction](#)  
[Append](#)  
[AppendRecord](#)  
[ApplyUpdates](#)  
[Assign](#)  
[BatchInput](#)  
[BatchOutput](#)  
[BeforeDestruction](#)  
[CachedUpdateStatus](#)  
[CancelUpdates](#)  
[CheckBrowseMode](#)  
[CheckNotUniDirectional](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[ClearFields](#)  
[Close](#)  
[CompareBookmarks](#)  
[ControlsDisabled](#)  
[Create](#)  
[CreateBlobStream](#)  
[CursorPosChanged](#)  
[DefaultHandler](#)  
[Delete](#)  
[Destroy](#)  
[DestroyComponents](#)  
[Destroying](#)  
[DisableControls](#)  
[Dispatch](#)  
[Edit](#)  
[EnableControls](#)  
[ExecuteAction](#)  
[FetchAll](#)  
[FieldAddress](#)  
[FieldByName](#)  
[FindComponent](#)  
[FindField](#)  
[FindFirst](#)

FindLast  
FindNext  
FindPrior  
First  
Free  
FreeBookmark  
FreeInstance  
FreeNotification  
FreeOnRelease  
GetBookmark  
GetCurrentRecord  
GetDetailDataSets  
GetDetailLinkFields  
GetFieldData  
GetFieldList  
GetFieldNames  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
GetNamePath  
GetParentComponent  
GetProviderAttributes  
GotoBookmark  
HasParent  
InheritsFrom  
InitInstance  
Insert  
InsertComponent  
InsertRecord  
InstanceSize  
IsEmpty  
IsLinkedTo  
Last  
Locate  
LocateNext  
Lookup  
MethodAddress  
MethodName  
MoveBy  
NewInstance  
Next  
Open  
Post  
Prior  
RecordModified



Refresh

RemoveComponent

Resync

RevertRecord

SafeCallException

SetFields

Translate

Undelete

UpdateAction

UpdateCursorPos

UpdateRecord

UpdateStatus

## TIBCustomDataSet.ApplyUpdates

[TIBCustomDataSet](#)

[See also](#)

Writes a dataset's pending cached updates to the database.

**procedure** ApplyUpdates;

### Description

Call ApplyUpdates to write a dataset's pending cached updates to a database. This method passes cached data to the database for storage, but the changes are not committed to the database. An application must explicitly call the TIBTransaction component's Commit method to commit the changes to the database if the write is successful, or call the TIBTransaction component's Rollback method to undo the changes if there is an error.

**Note:** The preferred method for updating datasets is to call a database component's ApplyUpdates method rather than to call each individual dataset's ApplyUpdates method. The application is responsible for committing or rolling back the transaction.

## **TIBCustomDataSet.BatchInput**

[TIBCustomDataSet](#)

[See also](#)

Executes the parameterized query in SQL for input in the referenced input object.

**procedure** BatchInput (InputObject: [TIBBatchInput](#));

### **Description**

Call BatchInput to execute the parameterized query in SQL for input in the referenced input object.

## **TIBCustomDataSet.BatchOutput**

[TIBCustomDataSet](#)

[See also](#)

Outputs the selected query in SQL to the referenced OutputObject.

**procedure** BatchOutput (OutputObject: [TIBBatchOutput](#));

### **Description**

Call BatchOutput to output the selected query in SQL to the referenced OutputObject.

## TIBCustomDataSet.CachedUpdateStatus

[TIBCustomDataSet](#)

[See also](#)

Returns the status of the cached updates.

```
type TCachedUpdateStatus = (cusUnmodified, cusModified, cusInserted,  
    cusDeleted, cusUninserted);
```

```
function CachedUpdateStatus: TCachedUpdateStatus;
```

### Description

Call CachedUpdateStatus to return the cached update status of the current record in the dataset.

TCachedUpdateStatus can be one of the following:

cusDeleted	Record will be deleted
cusInserted	Record will be inserted
cusModified	Record will be modified
cusUninserted	Record was inserted and then deleted
cusUnmodified	Record was not modified

## **TIBCustomDataSet.CancelUpdates**

[TIBCustomDataSet](#)

[See also](#)

Clears all pending cached updates from the cache.

**procedure** CancelUpdates;

### **Description**

Call CancelUpdates to clear all pending cached updates from the cache.

When a dataset is closed, or the CachedUpdates property is set to False, CancelUpdates is called automatically.

**Note:** To undo changes to a single record, call RevertRecord.

## TIBCustomDataSet.Create

[TIBCustomDataSet](#)

[See also](#)

Creates an instance of a TDataSet component.

**constructor** Create (AOwner: TComponent) ;

### Description

Call Create to instantiate a dataset component at runtime. Ordinarily applications instantiate dataset descendants, such as TIBTable, TIBQuery, TIBDataSet, or TIBStoredProc, rather than TIBCustomDataSet. These instantiated objects are handled automatically.

On the other hand, applications that create specialized dataset components, such as custom components, may need to instantiate a TIBCustomDataSet component by calling Create. Create:

- Calls the inherited Create for TDataSet.
- Creates the five query objects.
- Creates the base object to hold the reference to the database and transaction.
- Associates database and transaction related events to the base object component.

## TIBCustomDataSet.CreateBlobStream

[TIBCustomDataSet](#)

[See also](#)

[Example](#)

Returns a TBlobStream object for reading or writing the data in a specified blob field.

**type** TBlobStreamMode = (bmRead, bmWrite, bmReadWrite);

**function** CreateBlobStream(Field: TField; Mode: TBlobStreamMode) : TStream;

### Description

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



## TIBCustomDataSet.Destroy

[TIBCustomDataSet](#)

[See also](#)

Destroys the instance of a dataset component.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Usually destruction of datasets is handled automatically by Delphi. If an application creates its own instances of a dataset, however, the application should call Free, which verifies that the dataset component is not already freed before calling Destroy.

Destroy performs the following tasks:

- Frees the associated query objects.
- Frees the base object.
- Frees the associated internal resources.

## TIBCustomDataSet.FetchAll

[TIBCustomDataSet](#)

[See also](#)

Retrieves all records from the current cursor position to the end of the file and stores them locally.

**procedure** FetchAll;

### Description

Call FetchAll to reduce network traffic when using cached updates. FetchAll calls CheckBrowseMode to post any pending changes, and then retrieves all records from the current cursor position to the end of the file, and stores them locally. Ordinarily when cached updates are enabled, a transaction retrieves only as much data as it needs for return purposes.

**Note:** Using FetchAll is not always appropriate. For example, when an application accesses a database used by many simultaneous clients and there is a high degree of contention for updating the same records, fetching all records at once may not be advantageous because some fetched records may be changed by other applications. Always weigh the advantages of reduced network traffic against the need for reduced record contention.

## TIBCustomDataSet.GetCurrentRecord

[TIBCustomDataSet](#)

[See also](#)

Retrieves the current record into a buffer.

**function** GetCurrentRecord(Buffer: PChar) : Boolean;

### Description

Most applications should not need to call GetCurrentRecord. TDataSet automatically allocates a buffer for the active record.

Call GetCurrentRecord to copy the current record into a buffer allocated by the application. Buffer must be at least as big as the record size indicated by the RecordSize property.

## TIBCustomDataSet.GetFieldData

[TIBCustomDataSet](#)

[See also](#)

Reads the field data into a buffer.

**function** GetFieldData(FieldNo: Integer; Buffer: Pointer): Boolean;

**function** GetFieldData(Field: TField; Buffer: Pointer): Boolean;

### Description

GetFieldData reads field data from a field of a dataset specified by Field or FieldNo into a Buffer.  
Returns the size of the Buffer.

## TIBCustomDataSet.Locate

[TIBCustomDataSet](#)

[See also](#)

Searches the dataset for a specified record and makes that record the current record.

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

### Description

Call Locate to search a dataset for a specific record and position the cursor on it.

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine. For example:

```
with CustTable do  
Locate('Company;Contact;Phone', VarArrayOf(['Sight Diver', 'P', '408-431-  
1000']), [loPartialKey]);
```

Options is a set that optionally specifies additional search latitude when searching on string fields. If Options contains the loCaseInsensitive setting, then Locate ignores case when matching fields. If Options contains the loPartialKey setting, then Locate allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

Locate returns True if it finds a matching record, and makes that record the current one. Otherwise Locate returns False.

Locate uses the fastest possible method to locate matching records. If the search fields in KeyFields are indexed and the index is compatible with the specified search options, Locate uses the index. Otherwise Locate creates a filter for the search.

## TIBCustomDataSet.LocateNext

[TIBCustomDataSet](#)

[See also](#)

Searches the dataset for the record after a specified record and makes that record the current record.

```
function LocateNext(const KeyFields: string; const KeyValues: Variant;  
Options: TLocateOptions): Boolean
```

### Description

Call LocateNext to search a dataset for a record after the current cursor position

KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant that specifies the values to match in the key fields. If KeyFields lists a single field, KeyValues specifies the value for that field on the desired record. To specify multiple search values, pass a variant array as KeyValues, or construct a variant array on the fly using the VarArrayOf routine.

Options is a set that optionally specifies additional search latitude when searching on string fields. If Options contains the loCaseInsensitive setting, then LocateNext ignores case when matching fields. If Options contains the loPartialKey setting, then LocateNext allows partial-string matching on strings in KeyValues. If Options is an empty set, or if KeyFields does not include any string fields, Options is ignored.

LocateNext returns True if it finds a matching record, and makes that record the current one. Otherwise LocateNext returns False.

LocateNext uses the fastest possible method to locate matching records. If the search fields in KeyFields are indexed and the index is compatible with the specified search options, LocateNext uses the index. Otherwise LocateNext creates a filter for the search.

## TIBCustomDataSet.Lookup

[TIBCustomDataSet](#)

[See also](#)

Retrieves field values from a record that matches specified search values.

```
function Lookup(const KeyFields: string; const KeyValues: Variant; const  
    ResultFields: string): Variant;
```

### Description

Call Lookup to retrieve values for specified fields from a record that matches search criteria. KeyFields is a string containing a semicolon-delimited list of field names on which to search.

KeyValues is a variant array containing the values to match in the key fields. To specify multiple search values, pass KeyValues as a variant array as an argument, or construct a variant array on the fly using the VarArrayOf routine.

ResultFields is a string containing a semicolon-delimited list of field names whose values should be returned from the matching record.

Lookup returns a variant array containing the values from the fields specified in ResultFields.

Lookup uses the fastest possible method to locate matching records. If the search fields in KeyFields are indexed, Lookup uses the index. Otherwise Lookup creates a filter for the search.

## TIBCustomDataSet.RecordModified

[TIBCustomDataSet](#)

[See also](#)

Sets the record to modified or unmodified.

**procedure** RecordModified(Value: Boolean);

### Description

Call RecordModified to mark a record as modified or unmodified.

**Note:** This method is for internal use.



## TIBCustomDataSet.RevertRecord

[TIBCustomDataSet](#)

[See also](#)

[Example](#)

Restores the current record in the dataset to an unmodified state when cached updates are enabled.

**procedure** RevertRecord;

### Description

Call RevertRecord to undo changes made to the current record when cached updates are enabled.

**Note:** To undo all changes to all pending updates in the cache, call CancelUpdates.

## **TIBCustomDataSet.Undelete**

[TIBCustomDataSet](#)

[See also](#)

Restores a record deleted from the dataset.

**procedure** Undelete;

### **Description**

Call Undelete to restore a record deleted or uninserted from the dataset.

## TIBCustomDataSet.UpdateStatus

[TIBCustomDataSet](#)

[See also](#)

Reports the update status for the current record.

**type** TUpdateStatus = (usUnmodified, usModified, usInserted, usDeleted);

**function** UpdateStatus: TUpdateStatus;

### Description

Call UpdateStatus to determine the update status for the current record when cached updates are enabled. Update status can change frequently as records are edited, inserted, or deleted. UpdateStatus offers a convenient method for applications to assess the current status before undertaking or completing operations that depend on the update status of records.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



TDataSet



## TIBDatabase

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBDatabase encapsulates an InterBase database connection.

### Unit

IBDatabase

### Description

Use TIBDatabase to encapsulate an InterBase database connection. All TIBCustomDataSet descendants and TIBSQL use the TIBDatabase component to gain access to databases.

## TIBDatabase properties

[TIBDatabase](#) [Alphabetically](#) [Legend](#)

### In TIBDatabase

- ▶ [DatabaseName](#)
- ▶ [DBParamByDPB](#)
- ▶
  - ▶ [DBSQLDialect](#)
- ▶ [DefaultTransaction](#)
- ▶ [Handle](#)
  - ▶ [HandleIsShared](#)
- ▶ [IdleTimer](#)
- ▶ [IsReadOnly](#)
- ▶ [Params](#)
- ▶ [SQLDialect](#)
- ▶ [SQLObjectCount](#)
  - ▶ [SQLObjects](#)
  - [TraceFlags](#)
- ▶ [TransactionCount](#)
  - ▶ [Transactions](#)

### Derived from TCustomConnection

[Connected](#)

- ▶ [DataSetCount](#)
  - ▶ [DataSets](#)
  - [LoginPrompt](#)

### Derived from TComponent

- ▶ [ComObject](#)
  - ▶ [ComponentCount](#)
  - [ComponentIndex](#)
  - ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
  - [DesignInfo](#)
- ▶
  - ▶ [Name](#)
  - ▶ [Owner](#)
- ▶
  - [Tag](#)
  - [VCLComObject](#)



## TIBDatabase properties

[TIBDatabase](#) [By object](#) [Legend](#)

- ▶ ComObject
  - ▶ ComponentCount
  - ▶ ComponentIndex
  - ▶ Components
  - ▶ ComponentState
  - ▶ ComponentStyle
  - ▶ Connected
  - ▶ DatabaseName
  - ▶ DataSetCount
  - ▶ DataSets
  - ▶ DBParamByDPB
  - ▶ DBSQLDialect
  - ▶ DefaultTransaction
  - ▶ DesignInfo
  - ▶ Handle
  - ▶ HandleIsShared
  - ▶ IdleTimer
- ▶ IsReadOnly
  - ▶ LoginPrompt
  - ▶ Name
  - ▶ Owner
  - ▶ Params
  - ▶ SQLDialect
  - ▶ SQLObjectCount
  - ▶ SQLObjects
  - ▶ Tag
  - ▶ TraceFlags
  - ▶ TransactionCount
- ▶ Transactions
  - ▶ VCLComObject

## TIBDatabase.DatabaseName

[TIBDatabase](#) [See also](#)

Specifies the name of the database to associate with this database component.

**property** DatabaseName: **String**;

### Description

Use DatabaseName to specify the name of the database to use with a database component. For local InterBase databases, this can be a filename.

To connect to an InterBase database on a remote server using TCP/IP the syntax is  
<server\_name>:<filename>.

To connect to an InterBase database on a remote server using NetBEUI, the syntax is: \  
\<server\_name>\<filename>.

To connect to an InterBase database on a remote server using SPX, the syntax is:  
<server\_name>@<filename>.

## TIBDatabase.DBParamByDPB

[TIBDatabase](#) [See also](#)

Specifies the name of the database to associate with this database component.

**property** DBParamByDPB: [**const** Idx: Integer]: **String**;

### Description

Use DBParamByDPB to inspect and set DPB parameters without looking at the Params string list.

For example,

```
DBParamByDPB[isc_dpb_user_name]
```

can be used to set and inspect the user name.

## **TIBDatabase.DBSQLDialect**

[TIBDatabase](#) [See also](#)

Returns the database SQL dialect.

**property** DBSQLDialect: Integer;

### **Description**

Use DBSQLDialect to get the database SQL dialect.

## TIBDatabase.DefaultTransaction

[TIBDatabase](#) [See also](#)

Sets or returns the default database transaction.

**property** DefaultTransaction: TIBTransaction;

### Description

Use DefaultTransaction to set or return the default database transaction.

A single database connection can manage one or more transactions. DefaultTransaction is a convenient way to specify a default transaction to a database connection.

## TIBDatabase.Handle

[TIBDatabase](#) [See also](#)

Specifies the InterBase API database handle.

**property** Handle: TISC\_DB\_Handle;

### Description

Use Handle to make calls directly to the InterBase API. Many of the InterBase API functions require a database handle as one of their arguments. Handle is assigned an initial value when a database is opened.

## TIBDatabase.HandleIsShared

[TIBDatabase](#) [See also](#)

Indicates whether or not a the handle is shared.

**property** HandleIsShared: Boolean;

### Description

Read HandleIsShared to determine if the handle to the database is shared.

## **TIBDatabase.IdleTimer**

[TIBDatabase](#) [See also](#)

Specifies how long the database should wait before disconnecting an idle connection.

**property** IdleTimer: Integer;

### **Description**

Use IdleTimer to indicate how long the database should wait before automatically terminating the connection.



## **TIBDatabase.IsReadOnly**

[TIBDatabase](#) [See also](#)

Indicates whether or not the database is set to read-only.

**property** IsReadOnly: Boolean;

### **Description**

Read IsReadOnly to determine if the database is read-only.

**Note:** Read-only databases are an InterBase 6 feature.

## TIBDatabase.Params

[TIBDatabase](#) [See also](#)

Specifies the database parameters to pass to the InterBase server.

**property** Params: TStrings;

### Description

Use Params to specify the database parameters to pass to the InterBase server.

Database parameters are passed to the server as text in order to establish the connection. For example:

```
user_name=sysdba  
password=masterkey  
sql_role_name=finance  
lc_ctype=WIN1252
```

For more information on character sets, refer to “Character Sets and Collation Orders” in the InterBase Language Reference.

For other information, refer to the InterBase API Guide.

## TIBDatabase.SQLDialect

[TIBDatabase](#) [See also](#)

Sets or returns the SQL dialect used by the client.

**property** SQLDialect: Integer;

### Description

Use the SQLDialect property to set or return the SQL dialect used by the client. If the connection is active, the SQLDialect property cannot be set to a value greater than the database SQL dialect. If the connection is inactive, then on connect an OnDialectDownGradeWarning event may be fired if the SQLDialect is greater than the database SQL dialect. In such a case, the SQLDialect property will be downgraded to match the database SQL dialect.

## **TIBDatabase.SQLObjectCount**

[TIBDatabase](#) [See also](#)

Returns the number of SQL objects.

**property** `SQLObjectCount: Integer;`

### **Description**

Use the `SQLObjectCount` property to return the number of SQL objects in the database.

SQL objects are usually defined as InterBase datasets, IBSQL, and Blobs.

## TIBDatabase.SQLObjects

[TIBDatabase](#) [See also](#)

Returns an SQL object.

**property** SQLObjects[Index: Integer]: TIBBase;

### Description

Use the SQLObjects property to return an SQL object based on its numeric index.

SQL objects are usually defined as InterBase datasets, IBSQL, and Blobs.

## TIBDatabase.TraceFlags

[TIBDatabase](#) [See also](#)

Specifies the database operations to track with the SQL Monitor at runtime.

### type

```
TTraceFlag = (tfQPrepare, tfQExecute, tfQFetch, tfError, tfStmt, tfConnect,
  tfTransact, tfBlob, tfService, tfMisc);
TTraceFlags = set of TTraceFlag;
property TraceFlags: TTraceFlags;
```

### Description

Use TraceFlags to specify which database operations the SQL Monitor should track in an application at runtime. TraceFlags is only meaningful for the SQL Monitor, which is provided to enable performance tuning and SQL debugging when working with remote SQL database servers.

**Note:** Normally trace options are set from the SQL Monitor rather than setting TraceFlags in application code.

The value of a session component's TraceFlags property determines the initial settings of the TraceFlags property for database components associated with the session.

The TTraceFlags type defines the individual values that can be included in the TraceFlags property. The following table summarizes those values:

tfQPrepare	Monitor Prepare statements.
tfQExecute	Monitor ExecSQL statements.
tfQFetch	Monitor Fetch statements.
tfError	Monitor server error messages. Such messages may include an error code.
tfStmt	Monitor all SQL statements.
tfConnect	Monitor database connect and disconnect operations, including allocation of connection handles, and freeing connection handles.
tfTransact	Monitor transaction statements, such as StartTransaction, Commit, and Rollback.
tfBlob	Monitor operations on blob data types.
tfService	Monitor services.
tfMisc	Monitor any statements not covered by other flag options.

Because TraceFlags is a set property, an application can specify different combinations of flags to monitor different combinations of statements. For example, the following statement limits monitoring to database connections and SQL statement preparation: `TraceFlags := [tfConnect, tfQPrepare];`

## TIBDatabase.TransactionCount

[TIBDatabase](#) [See also](#)

Returns the number of transactions associated with the TIBDatabase component.

**property** TransactionCount: Integer;

### Description

Use TransactionCount to return how many transactions are currently associated with the InterBase database component.

## TIBDatabase.Transactions

[TIBDatabase](#) [See also](#)

Specifies a transaction for the given index.

**property** Transactions [**Index:** Integer]: TIBTransaction;

### Description

Given an integer index, Transactions returns the transaction at the given index. This is used internally for broadcasting important messages to attached components.



## TIBDatabase events

[TIBDatabase](#) [Alphabetically](#) [Legend](#)

### In TIBDatabase

- ▶ [OnDialectDowngradeWarning](#)
- ▶ [OnIdleTimer](#)
- ▶ [OnLogin](#)

### Derived from TCustomConnection

- ▶ [AfterConnect](#)
- ▶ [AfterDisconnect](#)
- ▶ [BeforeConnect](#)
- ▶ [BeforeDisconnect](#)

**TIBDatabase events**

[TIBDatabase](#)   [By object](#)   [Legend](#)

- ▶ [AfterConnect](#)
- ▶ [AfterDisconnect](#)
- ▶ [BeforeConnect](#)
- ▶ [BeforeDisconnect](#)
- ▶ [OnDialectDowngradeWarning](#)
- ▶ [OnIdleTimer](#)
- ▶ [OnLogin](#)

## TIBDatabase.OnDialectDowngradeWarning

[TIBDatabase](#) [See also](#)

Occurs after the SQL dialect of the client connection is downgraded.

**property** OnDialectDowngradeWarning: TNotifyEvent;

### Description

Write an OnDialectDowngradeWarning event handler to take specific actions when the SQL dialect is downgraded.

For example, if the SQL dialect for your application is set to 3 and then a connection is made to a dialect 1 database, then the SQL dialect is downgraded to 1 and a OnDialectDowngradeWarning event is fired.

## TIBDatabase.OnIdleTimer

[TIBDatabase](#) [See also](#)

Occurs after a database connection times out.

**property** OnIdleTimer: TNotifyEvent;

### Description

Write an OnIdleTimer event handler to take specific actions when the connection times out in the time specified by IdleTimer.

## TIBDatabase.OnLogin

[TIBDatabase](#) [See also](#)

Occurs when an application connects to a database.

```
TDatabaseLoginEvent = procedure (Database: TIBDatabase; LoginParams:
    TStrings) of object;
property OnLogin: TDatabaseLoginEvent;
```

### Description

Write an OnLogin event handler to take specific actions when an application attempts to connect to a database. By default, a database login is required. The current USER\_NAME is read from the Params property, and a standard Login dialog box opens. The dialog prompts for a user name and password combination, and then uses the values entered by the user to set the USER\_NAME and PASSWORD values in the Params property. These values are then passed to the remote server.

Applications that provide alternative OnLogin event handlers must set the USER\_NAME and PASSWORD values in LoginParams. LoginParams is a temporary string list and is freed automatically when no longer needed.

## **TIBDatabase methods**

[TIBDatabase](#) [Alphabetically](#)

### **In TIBDatabase**

[AddTransaction](#)

[ApplyUpdates](#)

[Call](#)

[CheckActive](#)

[CheckDatabaseName](#)

[CheckInactive](#)

[CloseDataSets](#)

[Create](#)

[CreateDatabase](#)

[Destroy](#)

[DropDatabase](#)

[FindTransaction](#)

[ForceClose](#)

[GetFieldNames](#)

[GetTableNames](#)

[IndexOfDBConst](#)

[RemoveTransaction](#)

[RemoveTransactions](#)

[SetHandle](#)

[TestConnected](#)

### **Derived from TCustomConnection**

[AddDataSet](#)

[Close](#)

[DoConnect](#)

[DoDisconnect](#)

[GetConnectedt](#)

[GetDataSet](#)

[GetDataSetCount](#)

[Loaded](#)

[Open](#)

[RemoveDataSet](#)

[SendConnectEvent](#)

[SetConnected](#)

### **Derived from TComponent**

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetNamePath](#)

GetParentComponent

HasParent

InsertComponent

RemoveComponent

SafeCallException

UpdateAction

**Derived from TPersistent**

Assign

**Derived from TObject**

AfterConstruction

BeforeDestruction

ClassInfo

ClassName

ClassNames

ClassParent

ClassType

CleanupInstance

DefaultHandler

Dispatch

FieldAddress

Free

FreeInstance

GetInterface

GetInterfaceEntry

GetInterfaceTable

InheritsFrom

InitInstance

InstanceSize

MethodAddress

MethodName

NewInstance

## TIBDatabase methods

[TIBDatabase](#) [By object](#)

[AddDataSet](#)  
[AddTransaction](#)  
[AfterConstruction](#)  
[ApplyUpdates](#)  
[Assign](#)  
[BeforeDestruction](#)  
[Call](#)  
[CheckActive](#)  
[CheckDatabaseName](#)  
[CheckInactive](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[Close](#)  
[CloseDataSets](#)  
[Create](#)  
[CreateDatabase](#)  
[DefaultHandler](#)  
[Destroy](#)  
[DestroyComponents](#)  
[Destroying](#)  
[Dispatch](#)  
[DoConnect](#)  
[DoDisconnect](#)  
[DropDatabase](#)  
[ExecuteAction](#)  
[FieldAddress](#)  
[FindComponent](#)  
[FindTransaction](#)  
[ForceClose](#)  
[Free](#)  
[FreeInstance](#)  
[FreeNotification](#)  
[FreeOnRelease](#)  
[GetConnectedt](#)  
[GetDataSet](#)  
[GetDataSetCount](#)  
[GetFieldNames](#)  
[GetInterface](#)



GetInterfaceEntry  
GetInterfaceTable  
GetNamePath  
GetParentComponent  
GetTableNames  
HasParent  
IndexOfDBConst  
InheritsFrom  
InitInstance  
InsertComponent  
InstanceSize  
Loaded  
Open  
RemoveDataSet  
SendConnectEvent  
SetConnected  
MethodAddress  
MethodName  
NewInstance  
Open  
RemoveComponent  
RemoveDataSet  
RemoveTransaction  
RemoveTransactions  
SafeCallException  
SendConnectEvent  
SetConnected  
SetHandle  
TestConnected  
UpdateAction

## TIBDatabase.AddTransaction

[TIBDatabase](#) [See also](#)

Adds an association between the transaction component and the database component.

**function** AddTransaction(TR: TIBTransaction): Integer;

### Description

Call AddTransaction to add an association between the transaction component and the database component.

## TIBDatabase.ApplyUpdates

[TIBDatabase](#) [See also](#)

Posts pending cached updates for specified datasets to the database server.

**procedure** ApplyUpdates (**const** DataSets: **array of** TIBCustomDataSet);

### Description

Call ApplyUpdates to post pending cached updates for a specific set of open datasets to the database server. ApplyUpdates is only meaningful if the CachedUpdates property of a specified dataset is True.

DataSets is a list of dataset names specifying the datasets for which to post pending updates. DataSets need not list every currently open dataset. For each listed dataset ApplyUpdates calls the dataset's ApplyUpdates and CommitUpdates methods to post that dataset's pending cached updates.

## TIBDatabase.Call

[TIBDatabase](#) [See also](#)

Returns an error message based on the error code.

**function** Call (ErrCode: ISC\_STATUS; RaiseError: Boolean): ISC\_STATUS;

### Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning an error based on the value of RaiseError.

## **TIBDatabase.CheckActive**

[TIBDatabase](#) [See also](#)

Checks to see if the database connection is active.

**procedure** `CheckActive;`

### **Description**

Call CheckActive to return an error if the connection to a database server is inactive.

## **TIBDatabase.CheckDatabaseName**

[TIBDatabase](#) [See also](#)

Checks to see if the DatabaseName property is not empty.

**procedure** CheckDatabaseName;

### **Description**

Call CheckDatabaseName to check if the DatabaseName property is empty, and to return an error if it is.

## **TIBDatabase.CheckInactive**

[TIBDatabase](#) [See also](#)

Checks to see if the database connection is inactive.

**procedure** CheckInactive;

### **Description**

Call CheckInactive to return an error if the connection to a database server is active.

## TIBDatabase.CloseDataSets

[TIBDatabase](#) [See also](#)

Closes all datasets associated with the database component without disconnecting from the database server.

**procedure** CloseDataSets;

### Description

Call CloseDataSets to close all active datasets without disconnecting from the database server.

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



## TIBDatabase.Create

[TIBDatabase](#) [See also](#)

Creates an instance of a TIBDatabase component.

**constructor** `Create (AOwner: TComponent);`

### Description

Call Create to instantiate a database component at runtime. An application can create a database component in order to control the component's existence and set its properties and events.

Create instantiates a database component and creates an empty list of dataset components for the DataSets property and an empty string list for the Params property.

## TIBDatabase.CreateDatabase

[TIBDatabase](#) [See also](#)

Creates a database using Params.

**procedure** CreateDatabase;

### Description

Call CreateDatabase to create a database using Params as the rest of the CREATE DATABASE command.

For example, if you wanted to create a local InterBase database, you could do the following:

1. Set the database name to the drive, path, and filename of the database file.
1. Set Params to the parameter for the CREATE DATABASE statement:

```
USER "SYSDBA"
```

```
0 PASSWORD "MASTERKEY"
```

```
1 PAGE_SIZE 4096
```

1. Set the SQLDialect value.
1. Call the CreateDatabase method.

## **TIBDatabase.Destroy**

[TIBDatabase](#) [See also](#)

Destroys the instance of the database component.

**destructor** `Destroy;`

### **Description**

Do not call Destroy directly in an application. Instead, call Free, which verifies that the database component is not already freed before calling Destroy.

Destroy disconnects from the database server, if necessary. It then frees the string resources allocated for the Params and DataSets properties before calling its inherited destructor.

## **TIBDatabase.DropDatabase**

[TIBDatabase](#) [See also](#)

Drops a database.

**procedure** DropDatabase;

### **Description**

Call DropDatabase to drop a database, which removes the database file from the server.

## TIBDatabase.FindTransaction

[TIBDatabase](#) [See also](#)

Finds the index of a transaction.

**function** FindTransaction (TR: TIBTransaction): Integer;

### Description

Call FindTransaction to find the index of a specified transaction.

## TIBDatabase.ForceClose

[TIBDatabase](#) [See also](#)

Forces the database connection to close.

**procedure** ForceClose;

### Description

Use ForceClose to force the database connection to close.

**Note:** Forcing a database to close attempts to close the connection to the server. Even if the call fails, the database handle is reset to nil.

## TIBDatabase.GetFieldNames

[TIBDatabase](#) [See also](#)

Populates a list with the names of the fields in the table.

**procedure** GetFieldNames(**const** TableName: string; List: TStrings);

### Description

Call GetFieldNames to retrieve a list of fields in the associated table.

## TIBDatabase.GetTableNames

[TIBDatabase](#) [See also](#)

Populates a string list with the names of tables in the database.

**procedure** GetTableNames(List: TStrings; SystemTables: Boolean = False);

### Description

Call GetTableNames to retrieve a list of tables in the associated database.

List is the already-existing string list object into which the tables names are put.

Set SystemTables to indicate whether the list of table names should include the database's system tables.

```
IBDatabase1.GetTableNames(ListBox2.Items, False);
```

Note: Any contents already in the target string list object are eliminated and overwritten by the data produced by GetTableNames.



## TIBDatabase.IndexOfDBConst

[TIBDatabase](#) [See also](#)

Searches for the named parameter in the database parameters list.

**function** IndexOfDBConst (st: **String**): Integer;

### Description

Use IndexOfDBConst to locate a parameter in the database parameters list. IndexOfDBConst returns – 1 if the parameter is not found.

## **TIBDatabase.RemoveTransaction**

[TIBDatabase](#) [See also](#)

Disassociates a transaction from the database.

**procedure** RemoveTransaction (Idx: Integer);

### **Description**

Call RemoveTransaction to disassociate a specified transaction from the database.

## **TIBDatabase.RemoveTransactions**

[TIBDatabase](#) [See also](#)

Disassociates all transactions from the database.

**procedure** RemoveTransactions;

### **Description**

Call RemoveTransactions to disassociate all transactions from the database.

## **TIBDatabase.SetHandle**

[TIBDatabase](#) [See also](#)

Sets the handle for the database.

**procedure** SetHandle;

### **Description**

Call SetHandle to set the handle for the database.

## TIBDatabase.TestConnected

[TIBDatabase](#) [See also](#)

Tests whether a database is connected.

**procedure** TestConnected: Boolean;

### Description

Use TestConnected to determine whether a database is connected to the server. TestConnected returns True if the connection is good, and False if it is not.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



TCustomConnection



## TIBDataLink

[Hierarchy](#)   [Properties](#)   [Methods](#)   [See also](#)

TIBDataLink is a helper class used by data-aware objects to coordinate the actions of TIBDataSet and to respond to data events.

### Unit

IBCustomDataSet

### Description

Use TIBDataLink or one of its descendants in any data-aware object that implements a DataSource property to represent its link to a dataset or that needs to respond to data events.

The constructor of the data-aware object should call the constructor of TIBDataLink, and initialize any relevant properties. The data-aware object can then link to a TDataSource by using the DataSource property of the TIBDataLink.

Data-aware objects that link to a single field in a dataset should use a TFieldDataLink instead.

## TIBDataLink properties

[TIBDataLink](#)   [Alphabetically Legend](#)

### Derived from TDetailDataLink

▀ [DetailDataSet](#)

### Derived from TDataLink

▀ [Active](#)

[ActiveRecord](#)

[BufferCount](#)

▀ [DataSet](#)

[DataSource](#)

[DataSourceFixed](#)

▀ [Editing](#)

[ReadOnly](#)

▀ [RecordCount](#)

## TIBDataLink properties

[TIBDataLink](#)

[By object](#)

[Legend](#)

### ▸ Active

ActiveRecord

BufferCount



DataSet

DataSource

DataSourceFixed



DetailDataSet



Editing

ReadOnly

### ▸ RecordCount

## TIBDataLink methods

[TIBDataLink](#)   [Alphabetically Legend](#)

### In TIBDataLink

- [ActiveChanged](#)
- ▶ [CheckBrowseMode](#)
- [Create](#)
- [Destroy](#)
- ▶ [GetDetailDataSet](#)
- ▶ [RecordChanged](#)

### Derived from TDataLink

- [Edit](#)
- [ExecuteAction](#)
- [UpdateAction](#)
- [UpdateRecord](#)

### Derived from TPersistent

- [Assign](#)
- [GetNamePath](#)

### Derived from TObject

- [AfterConstruction](#)
- [BeforeDestruction](#)
- [ClassInfo](#)
- [ClassName](#)
- [ClassNamels](#)
- [ClassParent](#)
- [ClassType](#)
- [CleanupInstance](#)
- [DefaultHandler](#)
- [Dispatch](#)
- [FieldAddress](#)
- [Free](#)
- [FreeInstance](#)
- [GetInterface](#)
- [GetInterfaceEntry](#)
- [GetInterfaceTable](#)
- [InheritsFrom](#)
- [InitInstance](#)
- [InstanceSize](#)
- [MethodAddress](#)
- [MethodName](#)
- [NewInstance](#)
- [SafeCallException](#)

## TIBDataLink methods

[TIBDataLink](#)   [By object](#)   [Legend](#)

▶	<a href="#">ActiveChanged</a>
	<a href="#">AfterConstruction</a>
▶	<a href="#">Assign</a>
	<a href="#">BeforeDestruction</a>
▶	<a href="#">CheckBrowseMode</a>
	<a href="#">ClassInfo</a>
	<a href="#">ClassName</a>
	<a href="#">ClassNamels</a>
	<a href="#">ClassParent</a>
	<a href="#">ClassType</a>
	<a href="#">CleanupInstance</a>
	<a href="#">Create</a>
	<a href="#">DefaultHandler</a>
	<a href="#">Destroy</a>
	<a href="#">Dispatch</a>
	<a href="#">Edit</a>
	<a href="#">ExecuteAction</a>
	<a href="#">FieldAddress</a>
	<a href="#">Free</a>
	<a href="#">FreeInstance</a>
▶	<a href="#">GetDetailDataSet</a>
	<a href="#">GetInterface</a>
	<a href="#">GetInterfaceEntry</a>
	<a href="#">GetInterfaceTable</a>
	<a href="#">GetNamePath</a>
	<a href="#">InheritsFrom</a>
	<a href="#">InitInstance</a>
	<a href="#">InstanceSize</a>
	<a href="#">MethodAddress</a>
	<a href="#">MethodName</a>
	<a href="#">NewInstance</a>
▶	<a href="#">RecordChanged</a>
	<a href="#">SafeCallException</a>
	<a href="#">UpdateAction</a>
	<a href="#">UpdateRecord</a>

## TIBDataLink.ActiveChanged

[TIBDataLink](#) [See also](#)

Responds to changes in the Active property.

**procedure** ActiveChanged;

### Description

The ActiveChanged method defined by TIBDataLink merely provides an interface for a method that can respond to changes in the Active property. Derived objects that do not need to respond to such changes can allow the inherited method to ignore them.

## **TIBDataLink.CheckBrowseMode**

[TIBDataLink](#) [See also](#)

Indicates the dataset browse mode.

**procedure** `CheckBrowseMode;`

### **Description**

Call `CheckBrowseMode` to indicate the dataset browse mode.

## TIBDataLink.Create

[TIBDataLink](#) [See also](#)

Creates an instance of TIBDataLink.

**constructor** Create (ADataset: TIBCustomDataSet);

### Description

Create is called from the constructor of any data-aware object that uses a TIBDataLink to implement its DataSource property.

After calling the inherited constructor, Create initializes the BufferCount property to 1. Data-aware objects that use a TIBDataLink object to manage their link to a DataSource should change the BufferCount property to the number of records they represent, after calling the inherited constructor.



## TIBDataLink.Destroy

[TIBDataLink](#) [See also](#)

Destroys an instance of TIBDataLink.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead, call the Free method. Free verifies that the TIBDataLink object is not already freed and only then calls Destroy.

Before calling the inherited destructor, Destroy removes any reference to the TIBDataLink from the data source object.

The TIBDataLink object should be destroyed in the destructor of its Owner, where that Owner calls Create from its constructor.

## TIBDataLink.GetDetailDataSet

[TIBDataLink](#) [See also](#)

Returns dataset details.

**function** GetDetailDataSet: TDataSet;

### Description

Call GetDetailDataSet to return details of the dataset.

## TIBDataLink.RecordChanged

[TIBDataLink](#) [See also](#)

Indicates whether a record has changed.

**procedure** RecordChanged(Field: TField);

### Description

The RecordChanged method defined by TIBDataLink merely provides an interface for a method that can respond to changes to the contents of the current record. RecordChanged is called after changes have been posted to the current record in the dataset.

The Field parameter indicates which field of the current record has changed in value. If Field is nil, any number of fields within the current record may have changed.

Derived objects that do not need to respond to such changes can allow the inherited method to ignore them.

## Scope



Published

## Accessibility



Read-only

**Scope**



Protected

## Hierarchy

TObject



TPersistent



TDataLink



TDetailDataLink



## TIBDataSet

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBDataSet executes InterBase SQL statements.

### Unit

IBCustomDataSet

### Description

Use TIBDataSet to execute InterBase SQL statements. TIBDataSet is primarily intended for use with SQL SELECT statements. TIBDataSet buffers the result set, making it completely scrollable. Since TIBDataSet is a descendant of TDataSet, it works well with all data-aware components.

## TIBDataSet properties

[TIBDataSet](#) [Alphabetically](#) [Legend](#)

### In TIBDataSet

- ▶ [BufferChunks](#)
- ▶ [DeleteSQL](#)
- ▶ [InsertSQL](#)
- ▶ [ModifySQL](#)
- ▶
  - ▶ [Params](#)
  - ▶ [Prepared](#)
  - ▶ [QDelete](#)
  - ▶ [QInsert](#)
  - ▶ [QModify](#)
  - ▶ [QRefresh](#)
  - ▶ [QSelect](#)
- ▶ [RefreshSQL](#)
- ▶ [SelectSQL](#)
- ▶
  - ▶ [StatementType](#)

### Derived from TIBCustomDataSet

- ▶ [Database](#)
- ▶ [DBHandle](#)
- ▶ [Transaction](#)
- ▶ [TRHandle](#)
- ▶ [UpdateObject](#)
- ▶ [UpdateRecordTypes](#)
- ▶ [UpdatesPending](#)

### Derived from TDataSet

- ▶ [Active](#)
- ▶
  - ▶ [AggFields](#)
- ▶
  - ▶ [AutoCalcFields](#)
  - ▶ [Bof](#)
  - ▶ [Bookmark](#)
- ▶ [CachedUpdates](#)
- ▶ [DatasetField](#)
- ▶ [DataSource](#)
- ▶ [DefaultFields](#)
- ▶
  - ▶ [Designer](#)
  - ▶ [Eof](#)
  - ▶ [FieldCount](#)
  - ▶ [FieldDefList](#)
  - ▶ [FieldDefs](#)
  - ▶ [FieldList](#)
- ▶ [Fields](#)
- ▶
  - ▶ [FieldValues](#)
  - ▶ [Found](#)
  - ▶ [Modified](#)
- ▶ [ObjectView](#)
- ▶
  - ▶ [RecordCount](#)
  - ▶ [SparseArrays](#)
  - ▶ [State](#)

### Derived from TComponent

- ▶ [ComObject](#)
- ▶ [ComponentCount](#)





## TIBDataSet properties

[TIBDataSet](#)

[By object](#)

[Legend](#)

▶	<a href="#">Active</a>
▶	▶ <a href="#">AggFields</a>
▶	<a href="#">AutoCalcFields</a>
▶	▶ <a href="#">Bof</a>
▶	▶ <a href="#">Bookmark</a>
▶	<a href="#">BufferChunks</a>
▶	<a href="#">CachedUpdates</a>
▶	▶ <a href="#">ComObject</a>
▶	<a href="#">ComponentCount</a>
	<a href="#">ComponentIndex</a>
	▶ <a href="#">Components</a>
	▶ <a href="#">ComponentState</a>
	▶ <a href="#">ComponentStyle</a>
▶	<a href="#">Database</a>
	<a href="#">DatasetField</a>
▶	<a href="#">DataSource</a>
▶	<a href="#">DBHandle</a>
▶	▶ <a href="#">DefaultFields</a>
▶	<a href="#">DeleteSQL</a>
	▶ <a href="#">Designer</a>
	<a href="#">DesignInfo</a>
	▶ <a href="#">Eof</a>
	▶ <a href="#">FieldCount</a>
	▶ <a href="#">FieldDefList</a>
▶	<a href="#">FieldDefs</a>
	▶ <a href="#">FieldList</a>
	▶ <a href="#">Fields</a>
	▶ <a href="#">FieldValues</a>
	▶ <a href="#">Found</a>
▶	<a href="#">InsertSQL</a>
	▶ <a href="#">Modified</a>
▶	<a href="#">ModifySQL</a>
▶	<a href="#">Name</a>
▶	<a href="#">ObjectView</a>
	▶ <a href="#">Owner</a>
	▶ <a href="#">Params</a>
	▶ <a href="#">Prepared</a>
	▶ <a href="#">QDelete</a>
	▶ <a href="#">QInsert</a>
	▶ <a href="#">QModify</a>
	▶ <a href="#">QRefresh</a>
	▶ <a href="#">QSelect</a>
	▶ <a href="#">RecordCount</a>
▶	<a href="#">RefreshSQL</a>
▶	<a href="#">SelectSQL</a>
	▶ <a href="#">SparseArrays</a>
	▶ <a href="#">Statet</a>
	▶ <a href="#">StatementType</a>
▶	<a href="#">Tag</a>
▶	<a href="#">Transaction</a>
▶	<a href="#">TRHandle</a>

- ▶ UpdateObject
- ▶ UpdateRecordTypes
- ▶ UpdatesPending
- VCLComObject

## **TIBDataSet.BufferChunks**

[TIBDataSet](#)

Sets or returns the dataset buffer chunk size.

**property** BufferChunks: **Integer**;

### **Description**

Use BufferChunks to set or return the dataset buffer chunk size as the number of records in the chunk.

## **TIBDataSet.DeleteSQL**

[TIBDataSet](#)   [See also](#)

Holds the SQL statement used to delete rows from the dataset.

**property** DeleteSQL: TStrings;

### **Description**

Use DeleteSQL to delete rows in the dataset.

## TIBDataSet.InsertSQL

[TIBDataSet](#)   [See also](#)

Holds the SQL statement used to insert rows into the dataset.

**property** InsertSQL: TStrings;

### Description

Use InsertSQL to insert rows into the dataset.

## **TIBDataSet.ModifySQL**

[TIBDataSet](#)   [See also](#)

Provides the ability to access the SQL object encapsulating the ModifySQL statement.

**property** ModifySQL: TStrings;

### **Description**

Use ModifySQL to access the SQL object that encapsulates the ModifySQL statement.

## TIBDataSet.Params

[TIBDataSet](#)   [See also](#)

Provides the ability to specify values for a parameterized query.

**property** Params: TIBXSQlda;

### Description

Use Params to specify values for a parameterized query.

For example:

```
DataSet1.Params[0].AsInteger = 24  
DataSet1.Params.ByName['Field2'].AsString = 'foo'
```



## TIBDataSet.Prepared

[TIBDataSet](#)   [See also](#)   [Example](#)

Determines whether or not a set of dataset queries is prepared for execution.

**property** Prepared: Boolean;

### Description

Examine Prepared to determine if a set of queries is already prepared for execution. If Prepared is True, the set of queries is prepared, and if Prepared is False, the set of queries is not prepared. While a set of queries need not be prepared before execution, execution performance is enhanced if the set of queries is prepared beforehand, particularly if it is a parameterized set of queries that is executed more than once using the same parameter values.

## **TIBDataSet.QDelete**

[TIBDataSet](#)   [See also](#)

Provides the ability to directly access the SQL object encapsulating the DeleteSQL statement.

**property** QDelete: TIBSQL;

### **Description**

Use QDelete to access the SQL object which encapsulates the Delete SQL statement.

## **TIBDataSet.QInsert**

[TIBDataSet](#)   [See also](#)

Provides the ability to directly access the SQL object encapsulating the InsertSQL statement.

**property** QInsert: TIBSQL;

### **Description**

Use QInsert to access the SQL object that encapsulates the InsertSQL statement.

## **TIBDataSet.QModify**

[TIBDataSet](#)   [See also](#)

Provides the ability to directly access the SQL object encapsulating the ModifySQL statement.

**property** QModify: TIBSQL;

### **Description**

Use QModify to access the SQL object that encapsulates the ModifySQL statement.

## **TIBDataSet.QRefresh**

[TIBDataSet](#)   [See also](#)

Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

**property** QRefresh: TIBSQL;

### **Description**

Use QRefresh to access the SQL object that encapsulates the RefreshSQL statement.

## **TIBDataSet.QSelect**

[TIBDataSet](#)   [See also](#)

Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.

**property** QSelect: TIBSQL;

### **Description**

Use QSelect to access the SQL object that encapsulates the SelectSQL statement.

## TIBDataSet.RefreshSQL

[TIBDataSet](#)   [See also](#)

Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

**property** RefreshSQL: TStrings;

### Description

Use RefreshSQL to access the SQL object that encapsulates the RefreshSQL statement.

## **TIBDataSet.SelectSQL**

[TIBDataSet](#)   [See also](#)

Provides the ability to directly access the SQL object encapsulating the SelectSQL statement.

**property** SelectSQL: TStrings;

### **Description**

Use SelectSQL to access the SQL object that encapsulates the SelectSQL statement.



## TIBDataSet.StatementType

[TIBDataSet](#)   [See also](#)

Returns the statement type of the QSelect query.

```
type TIBSQLTypes = set of (SQLUnknown, SQLSelect, SQLInsert, SQLUpdate,
SQLDelete, SQLDDL, SQLGetSegment, SQLPutSegment, SQLExecProcedure,
SQLStartTransaction, SQLCommit, SQLRollback, SQLSelectForUpdate,
SQLSetGenerator);
```

```
property StatementType: TIBSQLTypes;
```

### Description

Use StatementType to determine the statement type of a QSelect query. TIBSQLTypes are:

SQLCommit	Commits an active transaction
SQLDDL	Executes a DDL statement
SQLDelete	Removes rows in a table or in the active set of a cursor
SQLExecProcedure	Calls a stored procedure
SQLGetSegment	Reads a segment from an open Blob
SQLInsert	Adds one or more new rows to a specified table
SQLPutSegment	Writes a Blob segment
SQLRollback	Restores the database to its state prior to the start of the current transaction
SQLSetForUpdate	Stored procedure is set for updating
SQLSetGenerator	Sets a new value for an existing generator
SQLSelect	Retrieves data from one or more tables
SQLStartTransaction	Starts a new transaction against one or more databases
SQLUnknown	Unknown SQL type
SQLUpdate	Changes data in all or part of an existing row in a table, view, or active set of a cursor

## TIBDataSet events

[TIBDataSet](#)   [Alphabetically Legend](#)

### In TIBDataSet

- ▶ [DatabaseDisconnected](#)
- ▶ [DatabaseDisconnecting](#)
- ▶ [DatabaseFree](#)

### Derived from TIBCustomDataSet

- ▶ [OnUpdateError](#)
- ▶ [OnUpdateRecord](#)

### Derived from TIBDataSet

- ▶ [AfterCancel](#)
- ▶ [AfterClose](#)
- ▶ [AfterDelete](#)
- ▶ [AfterEdit](#)
- ▶ [AfterInsert](#)
- ▶ [AfterOpen](#)
- ▶ [AfterPost](#)
- ▶ [AfterRefresh](#)
- ▶ [AfterScroll](#)

- ▶ BeforeCancel
- ▶ BeforeClose
- ▶ BeforeDelete
- ▶ BeforeEdit
- ▶ BeforeInsert
- ▶ BeforeOpen
- ▶ BeforePost
- ▶ BeforeRefresh
- ▶ BeforeScroll
- ▶ OnCalcFields
- ▶ OnDeleteError
- ▶ OnEditError
- ▶ OnEditError
- ▶ OnFilterRecord
- ▶ OnPostError

## TIBDataSet events

<u>TIBDataSet</u>	<u>By object</u>	<u>Legend</u>
▶	<u>AfterCancel</u>	
▶	<u>AfterClose</u>	
▶	<u>AfterDelete</u>	
▶	<u>AfterEdit</u>	
▶	<u>AfterInsert</u>	
▶	<u>AfterOpen</u>	
▶	<u>AfterPost</u>	
▶	<u>AfterRefresh</u>	
▶	<u>AfterScroll</u>	
▶	<u>BeforeCancel</u>	
▶	<u>BeforeClose</u>	
▶	<u>BeforeDelete</u>	
▶	<u>BeforeEdit</u>	
▶	<u>BeforeInsert</u>	
▶	<u>BeforeOpen</u>	
▶	<u>BeforePost</u>	
▶	<u>BeforeRefresh</u>	
▶	<u>BeforeScroll</u>	
▶	<u>DatabaseDisconnected</u>	
▶	<u>DatabaseDisconnecting</u>	
▶	<u>DatabaseFree</u>	
▶	<u>OnCalcFields</u>	
▶	<u>OnDeleteError</u>	
▶	<u>OnEditError</u>	
▶	<u>OnEditError</u>	
▶	<u>OnFilterRecord</u>	
▶	<u>OnPostError</u>	
▶	<u>OnUpdateError</u>	
▶	<u>OnUpdateRecord</u>	
▶	<u>TransactionEnded</u>	
▶	<u>TransactionEnding</u>	
▶	<u>TransactionFree</u>	

## **TIBDataSet.DatabaseDisconnected**

[TIBDataSet](#)   [See also](#)

Occurs after a database has been disconnected.

**property** DatabaseDisconnected: TNotifyEvent;

### **Description**

Occurs after a database has been disconnected.

## **TIBDataSet.DatabaseDisconnecting**

[TIBDataSet](#)   [See also](#)

Occurs while a database is being disconnected.

**property** DatabaseDisconnecting: TNotifyEvent;

### **Description**

Occurs while a database is being disconnected.

## **TIBDataSet.DatabaseFree**

[TIBDataSet](#)   [See also](#)

Occurs after a database component is freed from memory.

**property** DatabaseFree: TNotifyEvent;

### **Description**

Occurs after a database component is freed from memory.

## **TIBDataSet.TransactionEnded**

[TIBDataSet](#)   [See also](#)

Occurs after a transaction has ended.

**property** TransactionEnded: TNotifyEvent;

### **Description**

Occurs after a transaction has ended.

## TIBDataSet.TransactionEnding

[TIBDataSet](#)   [See also](#)

Occurs before a transaction ends.

**property** TransactionEnding: TNotifyEvent;

### Description

Occurs before a transaction ends.



## **TIBDataSet.TransactionFree**

[TIBDataSet](#)   [See also](#)

Occurs after a transaction is freed from memory.

**property** TransactionFree: TNotifyEvent;

### **Description**

Occurs after a transaction is freed from memory.

## **TIBDataSet methods**

[TIBDataSet](#)   [Alphabetically](#)

### **In TIBDataSet**

[Prepare](#)

[UnPrepare](#)

### **Derived from TIBCustomDataSet**

[ApplyUpdates](#)

[CachedUpdateStatus](#)

[CancelUpdates](#)

[Create](#)

[CreateBlobStream](#)

[Destroy](#)

[FetchAll](#)

[GetCurrentRecord](#)

[GetFieldData](#)

[Locate](#)

[LocateNext](#)

[Lookup](#)

[RecordModified](#)

[RevertRecord](#)

[Undelete](#)

[UpdateStatus](#)

### **Derived from TDataSet**

[ActiveBuffer](#)

[Append](#)

[AppendRecord](#)

[CheckBrowseMode](#)

[ClearFields](#)

[Close](#)

[CompareBookmarks](#)

[ControlsDisabled](#)

[CursorPosChanged](#)

[Delete](#)

[DisableControls](#)

[Edit](#)

[EnableControls](#)

[FieldByName](#)

[FindField](#)

[FindFirst](#)

[FindLast](#)

[FindNext](#)

[FindPrior](#)

[First](#)

[FreeBookmark](#)

GetBookmark  
GetDetailDataSets  
GetDetailLinkFields  
GetFieldList  
GetFieldNames  
GetProviderAttributes  
GotoBookmark  
Insert  
InsertRecord  
IsEmpty  
IsLinkedTo  
Last  
MoveBy  
Next  
Open  
Post  
Prior  
Refresh  
Resync  
SetFields  
Translate  
UpdateCursorPos  
UpdateRecord

**Derived from TComponent**

DestroyComponents  
Destroying  
ExecuteAction  
FindComponent  
FreeNotification  
FreeOnRelease  
GetNamePath  
GetParentComponent  
HasParent  
InsertComponent  
RemoveComponent  
SafeCallException  
UpdateAction

**Derived from TPersistent**

Assign

**Derived from TObject**

AfterConstruction  
BeforeDestruction  
ClassInfo  
ClassName

ClassNamels  
ClassParent  
ClassType  
CleanupInstance  
DefaultHandler  
Dispatch  
FieldAddress  
Free  
FreeInstance  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
InheritsFrom  
InitInstance  
InstanceSize  
MethodAddress  
MethodName  
NewInstance

## TIBDataSet methods

[TIBDataSet](#)   [By object](#)

[ActiveBuffer](#)  
[AfterConstruction](#)  
[Append](#)  
[AppendRecord](#)  
[ApplyUpdates](#)  
[Assign](#)  
[BeforeDestruction](#)  
[CachedUpdateStatus](#)  
[CancelUpdates](#)  
[CheckBrowseMode](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[ClearFields](#)  
[Close](#)  
[CompareBookmarks](#)  
[ControlsDisabled](#)  
[Create](#)  
[CreateBlobStream](#)  
[CursorPosChanged](#)  
[DefaultHandler](#)  
[Delete](#)  
[Destroy](#)  
[DestroyComponents](#)  
[Destroying](#)  
[DisableControls](#)  
[Dispatch](#)  
[Edit](#)  
[EnableControls](#)  
[ExecuteAction](#)  
[FetchAll](#)  
[FieldAddress](#)  
[FieldByName](#)  
[FindComponent](#)  
[FindField](#)  
[FindFirst](#)  
[FindLast](#)  
[FindNext](#)  
[FindPrior](#)

First  
Free  
FreeBookmark  
FreeInstance  
FreeNotification  
FreeOnRelease  
GetBookmark  
GetCurrentRecord  
GetDetailDataSets  
GetDetailLinkFields  
GetFieldData  
GetFieldList  
GetFieldNames  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
GetNamePath  
GetParentComponent  
GetProviderAttributes  
GotoBookmark  
HasParent  
InheritsFrom  
InitInstance  
Insert  
InsertComponent  
InsertRecord  
InstanceSize  
IsEmpty  
IsLinkedTo  
Last  
Locate  
LocateNext  
Lookup  
MethodAddress  
MethodName  
MoveBy  
NewInstance  
Next  
Open  
Post  
Prepare  
Prior  
RecordModified  
Refresh  
RemoveComponent

Resync

RevertRecord

SafeCallException

SetFields

Translate

UpdateCursorPos

UpdateRecord

Undelete

UnPrepare

UpdateAction

UpdateStatus

## TIBDataSet.Prepare

[TIBDataSet](#)   [See also](#)   [Example](#)

Prepares all queries in the dataset to be executed.

**procedure** Prepare;

### Description

Call Prepare to prepare all queries in the dataset to be executed.



## **TIBDataSet.UnPrepare**

[TIBDataSet](#)   [See also](#)

Resets the state of a dataset's internal queries.

**procedure** UnPrepare;

### **Description**

Call UnPrepare to reset the state of a dataset's internal queries.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



TDataSet



TIBCustomDataSet

## TIBDataSetUpdateObject

[Hierarchy](#)   [Properties](#)   [Methods](#)   [See also](#)

TIBDataSetUpdateObject is the abstract base class for update objects used to update otherwise un-updateable queries when cached updates are enabled.

### Unit

IBCustomDataSet

### Description

Use TIBDataSetUpdateObject as a base class when creating customized update objects that can be used to update datasets. TIBDataSetUpdateObject declares a single property and some abstract methods, but provides no implementation details: these must be provided by descendant objects.

## TIBDataSetUpdateObject properties

[TIBDataSetUpdateObject](#)

[Alphabetically](#) [Legend](#)

### In TIBDataSetUpdateObject

- ▶ [DataSet](#)
- ▶ [RefreshSQL](#)

### Derived from TComponent

- ▶ [ComObject](#)
  - ▶ [ComponentCount](#)
  - ▶ [ComponentIndex](#)
  - ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
  - ▶ [DesignInfo](#)
- ▶ [Name](#)
- ▶ [Owner](#)
- ▶ [Tag](#)
- ▶ [VCLComObject](#)

TIBDataSetUpdateObject properties

[TIBDataSetUpdateObject](#)   [By object](#)   [Legend](#)

▶	<u>ComObject</u>	
▶		<u>ComponentCount</u>
		<u>ComponentIndex</u>
▶		<u>Components</u>
▶		<u>ComponentState</u>
▶		<u>ComponentStyle</u>
▶		<u>DataSet</u>
	<u>DesignInfo</u>	
▶		<u>Name</u>
	▶	<u>Owner</u>
▶		<u>Tag</u>
		<u>VCLComObject</u>

## **TIBDataSetUpdateObject.DataSet**

[TIBDataSetUpdateObject](#)   [See also](#)

Identifies the dataset to which a TIBDataSetUpdateObject component belongs.

**property** DataSet: TIBCustomDataSet;

### **Description**

Descendants of TIBDataSetUpdateObject must implement the abstract GetDataSet and SetDataSet methods to implement the DataSet property.



## **TIBDataSetUpdateObject.RefreshSQL**

[TIBDataSetUpdateObject](#)   [See also](#)

Provides the ability to directly access the SQL object encapsulating the RefreshSQL statement.

**property** RefreshSQL: TIBSQL;

### **Description**

Use RefreshSQL to access the SQL object that encapsulates the RefreshSQL statement.

## TIBDataSetUpdateObject methods

[TIBDataSetUpdateObject](#)   [Alphabetically](#)

[Legend](#)

### In TIBDataSetUpdateObject

- ▶ [Apply](#)
- ▶ [GetDataSet](#)
- ▶ [SetDataSet](#)

### Derived from TComponent

[DestroyComponents](#)  
[Destroying](#)  
[ExecuteAction](#)  
[FindComponent](#)  
[FreeNotification](#)  
[FreeOnRelease](#)  
[GetParentComponent](#)  
[HasParent](#)  
[InsertComponent](#)  
[RemoveComponent](#)  
[SafeCallException](#)  
[UpdateAction](#)

### Derived from TPersistent

[Assign](#)  
[GetNamePath](#)

### Derived from TObject

[AfterConstruction](#)  
[BeforeDestruction](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[DefaultHandler](#)  
[Dispatch](#)  
[FieldAddress](#)  
[Free](#)  
[FreeInstance](#)  
[GetInterface](#)  
[GetInterfaceEntry](#)  
[GetInterfaceTable](#)  
[InheritsFrom](#)  
[InitInstance](#)  
[InstanceSize](#)  
[MethodAddress](#)  
[MethodName](#)

NewInstance

## TIBDataSetUpdateObject methods

[TIBDataSetUpdateObject](#)

[By object](#)

[Legend](#)

 [Apply](#)  
[AfterConstruction](#)  
[Assign](#)  
[BeforeDestruction](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNames](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[DefaultHandler](#)  
[DestroyComponents](#)  
[Destroying](#)  
[Dispatch](#)  
[ExecuteAction](#)  
[FieldAddress](#)  
[FindComponent](#)  
[Free](#)  
[FreeInstance](#)  
[FreeNotification](#)  
[FreeOnRelease](#)  
 [GetDataSet](#)  
[GetInterface](#)  
[GetInterfaceEntry](#)  
[GetInterfaceTable](#)  
[GetNamePath](#)  
[GetParentComponent](#)  
[HasParent](#)  
[InheritsFrom](#)  
[InitInstance](#)  
[InsertComponent](#)  
[InstanceSize](#)  
[MethodAddress](#)  
[MethodName](#)  
[NewInstance](#)  
[RemoveComponent](#)  
[SafeCallException](#)  
 [SetDataSet](#)  
[UpdateAction](#)

## **TIBDataSetUpdateObject.Apply**

[TIBDataSetUpdateObject](#)   [See also](#)

Apply applies changes to the dataset specified by the DataSet property.

**type** TUpdateKind = (ukModify, ukInsert, ukDelete)

**procedure** Apply(UpdateKind: TUpdateKind);

### **Description**

Descendants of TIBDataSetUpdateObject must implement the abstract Apply method. This method is intended to perform the updates specified by the update object. Descendants must introduce properties to describe the details of the updates that should be performed. The UpdateKind parameter indicates whether the update object should modify existing records, insert new records, or delete existing records.

## **TIBDataSetUpdateObject.GetDataSet**

[TIBDataSetUpdateObject](#)    [See also](#)

GetDataSet returns the value of the DataSet property.

**function** GetDataSet: TIBCustomDataSet;

### **Description**

Override GetDataSet, along with the SetDataSet method, to provide an implementation of the DataSet property. The DataSet property should represent the dataset whose records are updated using the update object.

## **TIBDataSetUpdateObject.SetDataSet**

[TIBDataSetUpdateObject](#)    [See also](#)

SetDataSet sets the value of the DataSet property.

**procedure** SetDataSet (ADataset: TIBCustomDataSet) ;

### **Description**

Override SetDataSet, along with the GetDataSet method, to provide an implementation of the DataSet property. The DataSet property should represent the dataset whose records are updated using the update object.

## Scope



Protected  
Published  
Read-only



## Scope



Read-only

**Scope**



Protected

## Hierarchy

TObject



TPersistent



TComponent



## TIBDatabaseInfo

[Hierarchy](#)

[Properties](#)

[Methods](#)

TIBDatabaseInfo returns information about the attached database.

### Unit

IBDatabaseInfo

### Description

Use a TIBDatabaseInfo to return information about the attached database, such as the version of the online disk structure (ODS) used by the attachment, the number of database cache buffers allocated, the number of database pages read from or written to, or write-ahead log information.

## TIBDatabaseInfo properties

[TIBDatabaseInfo](#)

[Alphabetically](#) [Legend](#)

### In TIBDatabaseInfo

#### ▸ Allocation

- BackoutCount
- BaseLevel
- CurrentMemory
- Database
- DBFileName
- DBImplementationClass
- DBImplementationNo
- DBSiteName
- DBSQLDialect
- DeleteCount
- ExpungeCount

#### ▸ Fetches

- ForcedWrites
- InsertCount
- Marks
- MaxMemory
- NoReserve
- NumBuffers
- ODSMajorVersion
- ODSMinorVersion
- PageSize
- PurgeCount
- ReadIdxCount
- ReadOnly

#### ▸ Reads

- ReadSeqCount
- SweepInterval
- UpdateCount
- UserNames
- Version
- Writes

### Derived from TComponent

#### ▸ ComObject

- ComponentCount
- ComponentIndex
- Components
- ComponentState
- ComponentStyle
- DesignInfo
- Name
- Owner
- Tag
- VCLComObject

## TIBDatabaseInfo properties

[TIBDatabaseInfo](#)

[By object](#)

[Legend](#)

### ► [Allocation](#)

- [BackoutCount](#)
- [BaseLevel](#)
- [ComObject](#)
- [ComponentCount](#)
- [ComponentIndex](#)
- [Components](#)
- [ComponentState](#)

### ► [ComponentStyle](#)

- [CurrentMemory](#)
- [Database](#)
- [DBFileName](#)
- [DBImplementationClass](#)
- [DBImplementationNo](#)
- [DBSiteName](#)
- [DBSQLDialect](#)
- [DeleteCount](#)
- [DesignInfo](#)
- [ExpungeCount](#)

### ► [Fetches](#)

- [ForcedWrites](#)
- [InsertCount](#)
- [Marks](#)
- [MaxMemory](#)
- [Name](#)
- [NoReserve](#)
- [NumBuffers](#)
- [ODSMajorVersion](#)
- [ODSMinorVersion](#)
- [Owner](#)
- [PageSize](#)

### ► [PurgeCount](#)

- [ReadIdxCount](#)
- [ReadOnly](#)
- [Reads](#)
- [ReadSeqCount](#)
- [SweepInterval](#)
- [Tag](#)
- [UpdateCount](#)
- [UserNames](#)
- [VCLComObject](#)
- [Version](#)
- [Writes](#)

## **TIBDatabaseInfo.Allocation**

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of database pages allocated.

**property** Allocation: Long;

### **Description**

Use Allocation to return the number of database pages allocated.

## TIBDatabaseInfo.BackoutCount

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of removals of a version of a record.

**property** BackoutCount: TStringList;

### Description

Use BackoutCount to determine the number of times a version of a database record has been removed.



## TIBDatabaseInfo.BaseLevel

[TIBDatabaseInfo](#)

[See also](#)

Returns the database version number.

**property** BaseLevel: Long;

### Description

Use BaseLevel to return the database version number, which consists of 1 byte containing the number 1, and 1 byte containing the version number.

## **TIBDatabaseInfo.CurrentMemory**

[TIBDatabaseInfo](#)

[See also](#)

Returns the amount of server memory currently in use.

**property** CurrentMemory: Long;

### **Description**

Use CurrentMemory to return the amount of server memory (in bytes) currently in use.

## TIBDatabaseInfo.Database

[TIBDatabaseInfo](#)

[See also](#)

Sets or returns the database.

**property** Database: TIBDatabase;

### Description

Use Database to set or return the database on which information is being returned.

## TIBDatabaseInfo.DBFileName

[TIBDatabaseInfo](#)

[See also](#)

Returns the database filename.

**property** DBFileName: String;

### Description

Use DBFileName to return the database filename.

## TIBDatabaseInfo.DBImplementationClass

[TIBDatabaseInfo](#)

[See also](#)

Returns the database implementation class number.

**property** DBImplementationClass: Long;

### Description

Use DBImplementationClass to return the database implementation class number, either 1 or 12.

## TIBDatabaseInfo.DBImplementationNo

[TIBDatabaseInfo](#)

[See also](#)

Returns the database implementation number.

**property** DBImplementationNo: Long;

### Description

Use DBImplementationNo to return the database implementation number.

## TIBDatabaseInfo.DBSiteName

[TIBDatabaseInfo](#)

[See also](#)

Returns the database site name.

**property** DBSiteName: String;

### Description

Use DBSiteName to return the database site name.

## TIBDatabaseInfo.DBSQLDialect

[TIBDatabaseInfo](#)

[See also](#)

Returns the SQL dialect.

**property** DBSQLDialect: Long;

### Description

Use DBSQLDialect to return the SQL dialect.



## TIBDatabaseInfo.DeleteCount

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of database deletes since the database was last attached.

**property** DeleteCount: TStringList;

### Description

Use DeleteCount to return the number of database deletes since the database was last attached.

## TIBDatabaseInfo.ExpungeCount

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of removals of a record and all of its ancestors.

**property** ExpungeCount: TStringList;

### Description

Use ExpungeCount to return the number of removals of a record and all of its ancestors for records whose deletions have been committed.

## TIBDatabaseInfo.Fetches

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of reads from the memory buffer cache.

**property** Fetches: Long;

### Description

Use Fetches to return the number of reads from the memory buffer cache.

## **TIBDatabaseInfo.ForcedWrites**

[TIBDatabaseInfo](#)

[See also](#)

Returns the mode in which database writes are performed.

**property** ForcedWrites: Long;

### **Description**

Use ForcedWrites to return the number specifying the mode in which database writes are performed. ForcedWrites returns 0 for asynchronous mode, or returns 1 for synchronous mode

## TIBDatabaseInfo.InsertCount

[TIBDatabaseInfo](#)

[See also](#)

Returns number of inserts into the database since the database was last attached.

**property** InsertCount: TStringList;

### Description

Use InsertCount to return the number of inserts into the database since the database was last attached.

## TIBDatabaseInfo.Marks

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of writes to the memory buffer cache.

**property** Marks: Long;

### Description

Use Marks to return the number of writes to the memory buffer cache.

## TIBDatabaseInfo.MaxMemory

[TIBDatabaseInfo](#)

[See also](#)

Returns the maximum amount of memory used at one time since the first process attached to the database.

**property** MaxMemory: Long;

### Description

Use MaxMemory to return in bytes the maximum amount of memory used at one time since the first process attached to the database.

## TIBDatabaseInfo.NoReserve

[TIBDatabaseInfo](#)

[See also](#)

Returns whether or not space is reserved on each database page for holding backup versions of modified records.

**property** NoReserve: Long;

### Description

Use NoReserve to return whether or not space is reserved on each database page for holding backup versions of modified records. NoReserve will return 0 to indicate that space is reserved (the default) or 1 to indicate that no space is reserved.



## **TIBDatabaseInfo.NumBuffers**

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of memory buffers currently allocated.

**property** NumBuffers: Long;

### **Description**

Use NumBuffers to return the number of memory buffers currently allocated.

## TIBDatabaseInfo.ODSMajorVersion

[TIBDatabaseInfo](#)

[See also](#)

Returns the on disk structure (ODS) major version number.

**property** ODSMajorVersion: Long;

### Description

Use ODSMajorVersion to return the ODS major version number for the database. Databases with different major version numbers have different physical layouts.

A database engine can access only databases with a particular ODS major version number; trying to attach to a database with a different ODS number results in an error.

## TIBDatabaseInfo.ODSMajorVersion

[TIBDatabaseInfo](#)

[See also](#)

Returns the on disk structure (ODS) minor version number.

**property** ODSMinorVersion: Long;

### Description

Use ODSMinorVersion to return the (ODS) minor version number. An increase in a minor version number indicates a non-structural change, one that still allows the database to be accessed by databases with the same major version number but possibly different minor version numbers.

## TIBDatabaseInfo.PageSize

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of bytes per page of the attached database.

**property** PageSize: Long;

### Description

Use PageSize to return the number of bytes per page of the attached database. Use with Allocation to determine the size of the database.

## TIBDatabaseInfo.PurgeCount

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of removals of fully mature records from the database.

**property** PurgeCount: TStringList;

### Description

Use PurgeCount to return the number of removals of fully mature records (that is, records committed, resulting in older versions no longer being needed) from the database.

## TIBDatabaseInfo.ReadIdxCount

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of reads done via an index since the database was last attached.

**property** ReadIdxCount: TStringList;

### Description

Use ReadIdxCount to return the number of reads done via an index since the database was last attached.

## TIBDatabaseInfo.ReadOnly

[TIBDatabaseInfo](#)

[See also](#)

Indicates whether or not the database is read only.

**property** ReadOnly: Long;

### Description

Use ReadOnly to determine whether the database is read only or not. ReadOnly returns 1 if the database is read-write and 0 if it is read only.

**Note:** Read-only databases are an InterBase 6 feature.

## TIBDatabaseInfo.Reads

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of page reads from the database.

**property** Reads: Long;

### Description

Use Reads to return the number of page reads from the database since the current database was first attached, that is, an aggregate of all reads done by all attached processes, rather than the number of reads done for the calling program since it attached to the database.



## TIBDatabaseInfo.ReadSeqCount

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of sequential database reads done on each table since the database was last attached.

**property** ReadSeqCount: TStringList;

### Description

Use ReadSeqCount to return the number of sequential database reads (that is, the number of sequential table scans) done on each table since the database was last attached

## TIBDatabaseInfo.SweepInterval

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of transactions that are committed between “sweeps.”

**property** SweepInterval: Long;

### Description

Use SweepInterval to return the number of transactions that are committed between “sweeps” to remove database record versions that are no longer needed.

## TIBDatabaseInfo.UpdateCount

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of database updates since the database was last attached.

**property** UpdateCount: TStringList;

### Description

Use UpdateCount to return the number of database updates since the database was last attached.

## TIBDatabaseInfo.UserNames

[TIBDatabaseInfo](#)

[See also](#)

Returns the names of all users currently attached to the database.

**property** UserNames: TStringList;

### Description

Use UserNames to return the names of all users currently attached to the database.

## **TIBDatabaseInfo.Version**

[TIBDatabaseInfo](#)

[See also](#)

Returns the version of the database implementation.

**property** Version: String;

### **Description**

Use Version to return the version identification string of the database implementation.

## TIBDatabaseInfo.Writes

[TIBDatabaseInfo](#)

[See also](#)

Returns the number of page writes to the database.

**property** Writes: Long;

### Description

Use Writes to return the number of page writes to the current database since it was first attached by any process; that is, an aggregate of all write done by all attached processes, rather than the number of writes done for the calling program since it attached to the database.

## TIBDatabaseInfo methods

[TIBDatabaseInfo](#)

[Alphabetically](#)

### In TIBDatabaseInfo

[Call](#)

[Create](#)

[Destroy](#)

### Derived from TComponent

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### Derived from TPersistent

[Assign](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNames](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

NewInstance



## TIBDatabaseInfo methods

[TIBDatabaseInfo](#)

[By object](#)

[AfterConstruction](#)

[Assign](#)

[BeforeDestruction](#)

[Call](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecuteAction](#)

[FieldAddress](#)

[FindComponent](#)

[Free](#)

[FreeInstance](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

## TIBDatabaseInfo.Call

[TIBDatabaseInfo](#)

[See also](#)

Returns an error message based on the error code.

**function** Call(ErrCode: ISC\_STATUS; RaiseError: Boolean): ISC\_STATUS;

### Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning an error based on the value of RaiseError.

## TIBDatabaseInfo.Create

[TIBDatabaseInfo](#)

[See also](#)

Creates an instance of a DatabaseInfo component.

**constructor** Create (AOwner: TComponent) ;

### Description

Call Create to instantiate a DatabaseInfo component declared in an application.

## **TIBDatabaseInfo.Destroy**

[TIBDatabaseInfo](#)

[See also](#)

Destroys an instance of a DatabaseInfo component.

**Destructor** `Destroy;`

### **Description**

Do not call Destroy directly. Instead call Free to verify that the DatabaseInfo component is not already freed before calling Destroy. Destroy disconnects from the server, frees the parameter list, and calls its inherited Destroy destructor.

## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



TPersistent



TComponent

## **TIBDSBlobStream**

[Hierarchy](#)   [Properties](#)   [Methods](#)   [See also](#)

TIBDSBlobStream is an internal object used by the DataSet in the process of creating a Blob stream.

### **Unit**

IBCustomDataSet

### **Description**

TIBDSBlobStream is an internal object used by the DataSet in the process of creating a Blob stream.

## TIBDSBlobStream properties

[TIBDSBlobStream](#)

[Alphabetically Legend](#)

Derived from TStream

[Position](#)

[Size](#)



**TIBDSBlobStream properties**

[TIBDSBlobStream](#)

[By object](#)

[Legend](#)

[Position](#)

[Size](#)

## TIBDSBlobStream methods

[TIBDSBlobStream](#)

[Alphabetically](#)

### In TIBDSBlobStream

[Create](#)

[Read](#)

[Seek](#)

[SetSize](#)

[Write](#)

### Derived from TStream

[CopyFrom](#)

[ReadBuffer](#)

[ReadComponent](#)

[ReadComponentRes](#)

[ReadResHeader](#)

[WriteBuffer](#)

[WriteComponent](#)

[WriteComponentRes](#)

[WriteDescendent](#)

[WriteDescendentRes](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

## TIBDSBlobStream methods

[TIBDSBlobStream](#)

[By object](#)

[AfterConstruction](#)  
[BeforeDestruction](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[CopyFrom](#)  
[Create](#)  
[DefaultHandler](#)  
[Dispatch](#)  
[FieldAddress](#)  
[Free](#)  
[FreeInstance](#)  
[GetInterface](#)  
[GetInterfaceEntry](#)  
[GetInterfaceTable](#)  
[InheritsFrom](#)  
[InitInstance](#)  
[InstanceSize](#)  
[MethodAddress](#)  
[MethodName](#)  
[NewInstance](#)  
[Read](#)  
[ReadBuffer](#)  
[ReadComponent](#)  
[ReadComponentRes](#)  
[ReadResHeader](#)  
[Seek](#)  
[SetSize](#)  
[Write](#)  
[WriteBuffer](#)  
[WriteComponent](#)  
[WriteComponentRes](#)  
[WriteDescendent](#)  
[WriteDescendentRes](#)

## TIBDSBlobStream.Create

[TIBDSBlobStream](#)

[See also](#)

Resets the stream to the beginning of the Blob.

**type** TBlobStreamMode = (bmRead, bmWrite, bmReadWrite);

**constructor** Create(AField: TField ABlobStream: TIBBlobStream Mode: TBlobStreamMode);

### Description

Call Create to reset the stream to the beginning of the Blob. Depending on the mode, it also truncates the Blob stream.

## TIBDSBlobStream.Read

[TIBDSBlobStream](#)

[See also](#)

Reads the requested number of bytes from the Blob.

```
function Read(var Buffer; Count: Longint): Longint;
```

### Description

Call Read to read data from the Blob field when the number of bytes in the field's data is not known. Buffer must have at least Count bytes allocated to hold the data that was read from the field.

Read transfers up to Count bytes from the Blob data into Buffer, starting in the current position, and then advances the current position by the number of bytes actually transferred. Read returns the number of bytes actually transferred (which may be less than the number requested in Count.)

Read checks the Transliterate property of the field, and converts the data into ANSI from the character set specified by the dataset if Transliterate is True.

All the other data-reading methods of a Blob stream (ReadBuffer, ReadComponent) call Read to do their actual reading.

**Note:** Do not call Read when the TIBDSBlobStream was created in bmWrite mode.

## TIBDSBlobStream.Seek

[TIBDSBlobStream](#)

[See also](#)

Resets the current position of the TIBDSBlobStream object.

**function** Seek(Offset: Longint; Origin: Word): Longint;

### Description

Use Seek to move the current position within the Blob data by the indicated offset. Seek allows an application to read from or write to a particular location within the Blob data.

The Origin parameter indicates how to interpret the Offset parameter. Origin should be one of the following values:

Value	Meaning
soFromBeginning	<ul style="list-style-type: none"><li>Offset is from the beginning of the Blob data</li><li>Seek moves to the position Offset</li><li>Offset must be <math>\geq 0</math></li></ul>
soFromCurrent	<ul style="list-style-type: none"><li>Offset is from the current position in the Blob data</li><li>Seek moves to Position + Offset</li></ul>
soFromEnd	<ul style="list-style-type: none"><li>Offset is from the end of the Blob data</li><li>Offset must be <math>\leq 0</math> to indicate a number of bytes before the end of the Blob</li></ul>

Seek returns the new value of the Position property, the new current position in the Blob data.

## **TIBDSBlobStream.SetSize**

[TIBDSBlobStream](#)

[See also](#)

Set the size of the Blob to the requested size.

**procedure** SetSize (NewSize: Long) ;

### **Description**

Call SetSize to set the size of the Blob to the requested size.

## TIBDSBlobStream.Write

[TIBDSBlobStream](#)

[See also](#)

Sets the field to be modified.

```
function Write(const Buffer; Count: Longint): Longint;
```

### Description

Use Write to set the field to be modified, write the requested number of bytes to the Blob stream, and fire a OnFieldChange event.



## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



TStream

## **EIBError**

[Hierarchy](#)   [Properties](#)   [Methods](#)   [See also](#)

The exception class for InterBase errors.

### **Unit**

IB

### **Description**

Use EIBError to raise an exception when a component detects an error in the database or in the component implementation.

## EIBError properties

[EIBError](#)

[Alphabetically Legend](#)

### In EIBError

▸ [IBErrorCode](#)



[SQLCode](#)

### Derived from Exception

[HelpContext](#)

[Message](#)

EIBError properties

[EIBError](#)      [By object](#)      [Legend](#)

<u>HelpContext</u>	
▸ <u>IBErrorCode</u>	
▸	<u>Message</u>
	<u>SQLCode</u>

## **EIBError.IBErrorCode**

[EIBError](#)

[See also](#)

Returns the InterBase error code.

**property** IBErrorCode: Long;

### **Description**

Use IBErrorCode to get the InterBase error code.

## **EIBError.SQLCode**

[EIBError](#)      [See also](#)

Translates an InterBase error code in the error status vector to an SQL error number code.

**property** SQLCode: Long;

### **Description**

Use SQLCode to translate an InterBase error code in the error status vector to an SQL error number code. Typically, this call is used to populate a program variable with an SQL error number for use in an SQL error-handling routine.

## **EIBError methods**

[EIBError](#)      [Alphabetically](#)

### **In EIBError**

[Create](#)

### **Derived from Exception**

[CreateFmt](#)

[CreateFmtHelp](#)

[CreateHelp](#)

[CreateRes](#)

[CreateResFmt](#)

[CreateResFmtHelp](#)

[CreateResHelp](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)



## **EIBError methods**

[EIBError](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[CreateFmt](#)

[CreateFmtHelp](#)

[CreateHelp](#)

[CreateRes](#)

[CreateResFmt](#)

[CreateResFmtHelp](#)

[CreateResHelp](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## **EIBError.Create**

[EIBError](#)      [See also](#)

Calls the inherited Create and sets the SQLCSode and IBERrorCode

**constructor** Create (ASQLCode: **Long**; Msg: **string**);

**constructor** Create (ASQLCode: **Long**; AIBErrorCode: **Long**; Msg: **string**);

### **Description**

Use Create to call to the inherited Create and sets the SQL code and IBERrorCode.

## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



Exception



EDatabaseError

## **EIBClientError**

[Hierarchy](#)   [Properties](#)   [Methods](#)   [See also](#)

Used to raise client-specific errors.

### **Unit**

IB

### **Description**

Use EIBClientError to raise an exception for client-specific errors.

## EIBClientError properties

[EIBClientError](#) [Alphabetically](#) [Legend](#)

### Derived from EIBError

▸ [IBErrorCode](#)

▸ [SQLCode](#)

### Derived from Exception

[HelpContext](#)

[Message](#)

**EIBClientError properties**

[EIBClientError](#) [By object](#) [Legend](#)

<u>HelpContext</u>	
▀ <u>IBErrorCode</u>	
▀ <u>Message</u>	
▀ <u>SQLCode</u>	

## **EIBClientError methods**

[EIBClientError](#) [Alphabetically](#)

### **Derived from EIBError**

[Create](#)

### **Derived from Exception**

[CreateFmt](#)

[CreateFmtHelp](#)

[CreateHelp](#)

[CreateRes](#)

[CreateResFmt](#)

[CreateResFmtHelp](#)

[CreateResHelp](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)



## **EIBError methods**

[EIBClientError](#) [By object](#)

[AfterConstruction](#)  
[BeforeDestruction](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[Create](#)  
[CreateFmt](#)  
[CreateFmtHelp](#)  
[CreateHelp](#)  
[CreateRes](#)  
[CreateResFmt](#)  
[CreateResFmtHelp](#)  
[CreateResHelp](#)  
[DefaultHandler](#)  
[Dispatch](#)  
[FieldAddress](#)  
[Free](#)  
[FreeInstance](#)  
[GetInterface](#)  
[GetInterfaceEntry](#)  
[GetInterfaceTable](#)  
[InheritsFrom](#)  
[InitInstance](#)  
[InstanceSize](#)  
[MethodAddress](#)  
[MethodName](#)  
[NewInstance](#)  
[SafeCallException](#)

## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



Exception



EDatabaseError



EIBError

## EIBInterBaseError

[Hierarchy](#)   [Properties](#)   [Methods](#)   [See also](#)

Used to raise server-specific errors.

### Unit

IB

### Description

Use EIBInterBaseError to raise an exception for server-specific errors.

## EIBInterBaseError properties

[EIBInterBaseError](#)

[Alphabetically](#) [Legend](#)

### Derived from EIBError

▸ [IBErrorCode](#)



[SQLCode](#)

### Derived from Exception

[HelpContext](#)

[Message](#)

EIBInterBaseError properties

[EIBInterBaseError](#)      [By object](#)      [Legend](#)

<a href="#">HelpContext</a>	
▀ <a href="#">IBErrorCode</a>	
	<a href="#">Message</a>
▀	<a href="#">SQLCode</a>

## **EIBInterBaseError methods**

[EIBInterBaseError](#)

[Alphabetically](#)

### **Derived from EIBError**

[Create](#)

### **Derived from Exception**

[CreateFmt](#)

[CreateFmtHelp](#)

[CreateHelp](#)

[CreateRes](#)

[CreateResFmt](#)

[CreateResFmtHelp](#)

[CreateResHelp](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## EIBError methods

[EIBInterBaseError](#)

[By object](#)

[AfterConstruction](#)  
[BeforeDestruction](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[Create](#)  
[CreateFmt](#)  
[CreateFmtHelp](#)  
[CreateHelp](#)  
[CreateRes](#)  
[CreateResFmt](#)  
[CreateResFmtHelp](#)  
[CreateResHelp](#)  
[DefaultHandler](#)  
[Dispatch](#)  
[FieldAddress](#)  
[Free](#)  
[FreeInstance](#)  
[GetInterface](#)  
[GetInterfaceEntry](#)  
[GetInterfaceTable](#)  
[InheritsFrom](#)  
[InitInstance](#)  
[InstanceSize](#)  
[MethodAddress](#)  
[MethodName](#)  
[NewInstance](#)  
[SafeCallException](#)



## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



Exception



EDatabaseError



EIBError

## **TIBInputDelimitedFile**

[Hierarchy](#)   [Properties](#)   [Methods](#)[See also](#)

TIBInputDelimitedFile performs batch input of data from delimited files.

### **Unit**

IBBatch

### **Description**

Use a TIBInputDelimitedFile object to perform batch input of data from delimited files.

**TIBInputDelimitedFile properties**

[TIBInputDelimitedFile](#)      [Alphabetically Legend](#)

**In TIBInputDelimitedFile**

- [ColDelimiter](#)
- [ReadBlanksAsNull](#)
- [RowDelimiter](#)
- [SkipTitles](#)

**Derived from TIBBatch**

- ▶ [Columns](#)
  - ▶ [FileName](#)
  - ▶ [Params](#)

TIBInputDelimitedFile properties

[TIBInputDelimitedFile](#)   [By object](#)   [Legend](#)

<u>ColDelimiter</u>	
▸ <u>Columns</u>	
▸	<u>FileName</u>
	<u>Params</u>
	<u>ReadBlanksAsNull</u>
	<u>RowDelimiter</u>
	<u>SkipTitles</u>

## **TIBInputDelimitedFile.ColDelimiter**

[TIBInputDelimitedFile](#)

[See also](#)

Sets the column delimiter for the input file.

**property** ColDelimiter: **String**;

### **Description**

Use ColDelimiter to set the column delimiter (either Tab-Ctrl-F or |~)for the input file.

## TIBInputDelimitedFile.ReadBlanksAsNull

[TIBInputDelimitedFile](#)

[See also](#)

Reads blank spaces in the input file as null characters.

**property** ReadBlanksAsNull: Boolean;

### Description

Set ReadBlanksAsNull to True read blank spaces as null characters in the input file.

## TIBInputDelimitedFile.RowDelimiter

[TIBInputDelimitedFile](#)

[See also](#)

Sets the row delimiter for the input file.

**property** RowDelimiter: **String**;

### Description

Use RowDelimiter to set the column delimiter (either Tab-Ctrl-F or |~) for the input file.



## TIBInputDelimitedFile.SkipTitles

[TIBInputDelimitedFile](#)

[See also](#)

Skips the first record of a delimited file.

**property** SkipTitles: Boolean;

### Description

Set SkipTitles to True to treat the first record of a delimited file as titles and skip it. Field titles are not useful in batch inputs, and this property allows you to skip them.

## TIBInputDelimitedFile methods

[TIBBatchInput](#) [Alphabetically](#)

### In TIBInputDelimitedFile

[Destroy](#)

[GetColumn](#)

[ReadParameters](#)

[ReadyFile](#)

### Derived from TIBBatch

[Move](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBInputDelimitedFile methods

[TIBBatch](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Destroy](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetColumn](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[Move](#)

[NewInstance](#)

[ReadParameters](#)

[ReadyFile](#)

[SafeCallException](#)

## TIBInputDelimitedFile.Destroy

[TIBInputDelimitedFile](#)

[See also](#)

Destroys the instance of TIBInputDelimitedFile.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBInputDelimitedFile object is not already freed and only then calls Destroy.

## TIBInputDelimitedFile.GetColumn

[TIBInputDelimitedFile](#)

[See also](#)

Returns the contents of a column.

```
function GetColumn: var Col: String): Integer;
```

### Description

Call GetColumn to return the contents of a column in the extended SQL descriptor area (XSQLDA).

## **TIBInputDelimitedFile.ReadParameters**

[TIBInputDelimitedFile](#)

[See also](#)

Reads the input parameters of the XSQLDA.

**function** ReadParameters: Boolean;

### **Description**

Call ReadParameters to read the input parameters of the extended SQL descriptor area (XSQLDA).

## **TIBInputDelimitedFile.ReadyFile**

[TIBInputDelimitedFile](#)

[See also](#)

Prepares the output for the XSQLDA.

**procedure** ReadyFile;

### **Description**

Call ReadyFile to prepare the output for the extended SQL descriptor area (XSQLDA).

## Scope



Published

## Accessibility



Read-only



## Hierarchy

TObject



TIBBatch



TIBBatchInput

## TIBInputRawFile

[Hierarchy](#)   [Properties](#)   [MethodsSee also](#)

TIBInputRawFile inputs data from a raw file.

### Unit

IBBatch

### Description

Use a TIBInputRawFile object input data from a raw file. A raw file is the equivalent to InterBase external file output. Raw files are not limited to a straight character format.

**TIBInputRawFile properties**

[TIBInputRawFile](#)      [Alphabetically Legend](#)

**Derived from TIBBatch**

▶ <a href="#"><u>Columns</u></a>	
▶	<a href="#"><u>FileName</u></a>
	<a href="#"><u>Params</u></a>

TIBInputRawFile properties

[TIBInputRawFile](#)      [By object](#)      [Legend](#)

▸ [Columns](#)

▸ [FileName](#)  
[Params](#)

## TIBInputRawFile methods

[TIBInputRawFile](#)

[Alphabetically](#)

### In TIBInputRawFile

[Destroy](#)

[ReadParameters](#)

[ReadyFile](#)

### Derived from TIBBatch

[Move](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBInputRawFile methods

[TIBBatch](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Destroy](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[Move](#)

[NewInstance](#)

[ReadParameters](#)

[ReadyFile](#)

[SafeCallException](#)

## TIBInputRawFile.Destroy

[TIBInputRawFile](#)

[See also](#)

Destroys the instance of TIBInputRawFile.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBInputRawFile object is not already freed and only then calls Destroy.

## TIBInputRawFile.ReadParameters

[TIBInputRawFile](#)

[See also](#)

Reads the XSQLDA input parameters.

**function** ReadParameters: Boolean;

### Description

Call ReadParameters to read the input parameters of the extended SQL descriptor area (XSQLDA).



## **TIBInputRawFile.ReadyFile**

[TIBInputRawFile](#)

[See also](#)

Prepares the output for the XSQLDA.

**procedure** ReadyFile;

### **Description**

Call ReadyFile to prepare the output for the extended SQL descriptor area (XSQLDA).

## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



TIBBatch



TIBBatchInput

## **TIBOutputDelimitedFile**

[Hierarchy](#)   [Properties](#)   [Methods](#)[See also](#)

TIBOutputDelimitedFile performs batch output of data to delimited files.

### **Unit**

IBBatch

### **Description**

Use a TIBOutputDelimitedFile object to perform batch output of data to delimited files.

**TIBOutputDelimitedFile properties**

[TIBOutputDelimitedFile](#)      [Alphabetically Legend](#)

**In TIBOutputDelimitedFile**

ColDelimiter

OutputTitles

RowDelimiter

**Derived from TIBBatch**

▀ Columns

FileName

▀ Params

**TIBOutputDelimitedFile properties**

[TIBOutputDelimitedFile](#)    [By object](#)    [Legend](#)

	<u>ColDelimiter</u>
▸	<u>Columns</u>
	<u>FileName</u>
	<u>OutputTitles</u>
▸	<u>Params</u>
	<u>RowDelimiter</u>

## **TIBOutputDelimitedFile.ColDelimiter**

[TIBOutputDelimitedFile](#)

[See also](#)

Sets the column delimiter for the output file.

**property** ColDelimiter: **String**;

### **Description**

Use ColDelimiter to set the column delimiter (either Tab-Ctrl-F or |~)for the output file.

## **TIBOutputDelimitedFile.OutputTitles**

[TIBOutputDelimitedFile](#)      [See also](#)

Outputs the titles at the top of the file.

**property** OutputTitles: Boolean;

### **Description**

Set OutputTitles to True to output the titles at the top of the file.



## TIBOutputDelimitedFile.RowDelimiter

[TIBOutputDelimitedFile](#)

[See also](#)

Sets the row delimiter for the output file.

**property** RowDelimiter: **String**;

### Description

Use RowDelimiter to set the column delimiter (either Tab-Ctrl-F or |~) for the output file.

## TIBOutputDelimitedFile methods

[TIBBatchOutput](#)

[Alphabetically](#)

### In TIBOutputDelimitedFile

[Destroy](#)

[ReadyFile](#)

[WriteColumns](#)

### Derived from TIBBatch

[Move](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBOutputDelimitedFile methods

[TIBBatch](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Destroy](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[Move](#)

[NewInstance](#)

[ReadyFile](#)

[SafeCallException](#)

[WriteColumns](#)

## TIBOutputDelimitedFile.Destroy

[TIBOutputDelimitedFile](#)

[See also](#)

Destroys the instance of TIBOutputDelimitedFile.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBOutputDelimitedFile object is not already freed and only then calls Destroy.

## **TIBOutputDelimitedFile.ReadyFile**

[TIBOutputDelimitedFile](#)      [See also](#)

Prepares the output for the XSQLDA.

**procedure** ReadyFile;

### **Description**

Call Ready file to prepare the output for the extended SQL descriptor area (XSQLDA).

## TIBOutputDelimitedFile.WriteColumns

[TIBOutputDelimitedFile](#)      [See also](#)

Outputs the data in columns in the XSQLDA.

**function** WriteColumns: Boolean;

### Description

Set WriteColumns to True to output data in columns in the extended SQL descriptor area (XSQLDA).

## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



TIBBatch



TIBBatchOutput



## TIBOutputRawFile

[Hierarchy](#)   [Properties](#)   [Methods](#)[See also](#)

TIBOutputRawFile outputs data to a raw file format.

### Unit

IBBatch

### Description

Use a TIBOutputRawFile object to output data from a raw file. A raw file is the equivalent to InterBase external file output. Raw files are not limited to a straight character format.

**TIBOutputRawFile properties**

[TIBOutputRawFile](#)      [Alphabetically Legend](#)

**Derived from TIBBatch**

- ▶ [Columns](#)
  - ▶ [FileName](#)
  - ▶ [Params](#)

TIBOutputRawFile properties

[TIBOutputRawFile](#)

[By object](#)

[Legend](#)

▸ [Columns](#)



FileName

Params

## TIBOutputRawFile methods

[TIBBatchOutputAlphabetically](#)

### In TIBOutputRawFile

[Destroy](#)

[ReadyFile](#)

[WriteColumns](#)

### Derived from TIBBatch

[Move](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBOutputRawFile methods

[TIBBatch](#)

[By object](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Destroy](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[Move](#)

[NewInstance](#)

[ReadyFile](#)

[SafeCallException](#)

[WriteColumns](#)

## TIBOutputRawFile.Destroy

[TIBOutputRawFile](#)

[See also](#)

Destroys the instance of TIBOutputRawFile.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBOutputRawFile object is not already freed and only then calls Destroy.

## **TIBOutputRawFile.ReadyFile**

[TIBOutputRawFile](#)

[See also](#)

Prepares the output for the XSQLDA.

**procedure** ReadyFile;

### **Description**

Call Ready file to prepare the output for the extended SQL descriptor area (XSQLDA).

## **TIBOutputRawFile.WriteColumns**

[TIBOutputRawFile](#)

[See also](#)

Outputs the data in columns in the XSQLDA.

**function** WriteColumns: Boolean;

### **Description**

Call WriteColumns to output data in columns in the extended SQL descriptor area (XSQLDA).



## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



TIBBatch



TIBBatchOutput



## TIBQuery

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBQuery executes an InterBase SQL statement.

### Unit

IBQuery

### Description

Use TIBQuery to access one or more tables in a database using SQL statements. Use query components with remote InterBase database servers and with ODBC-compliant databases.

Query components are useful because they can

- Access more than one table at a time (called a “join” in SQL).
- Automatically access a subset of rows and columns in its underlying table(s), rather than always returning all rows and columns.

**Note:** TIBQuery is of particular importance to the development of scalable database applications. If there is any chance that an application built to run against local databases will be scaled to a remote SQL database server in the future, use TIBQuery components from the start to ensure easier scaling later.

## TIBQuery properties

[TIBQuery](#)    [Alphabetically Legend](#)

### In TIBQuery

#### GenerateParamNames

- ▶ ParamCheck
- ▶ ParamCount
- ▶ Params
- ▶ Prepared
- ▶ RowsAffected
- ▶ SQL
- ▶ StmtHandle
- ▶ Text
- ▶ UniDirectional
- ▶ UpdateObject

### Derived from TIBCustomDataSet

- ▶ CachedUpdates
- ▶ Database
- ▶ DBHandle
- ▶ Transaction
- ▶ TRHandle
- ▶ UpdateRecordTypes
- ▶ UpdatesPending

### Derived from TDataSet

- ▶ Active
- ▶ AggFields
- ▶ AutoCalcFields
- ▶ Bof
- ▶ Bookmark
- ▶ Constraints
- ▶ DatasetField
- ▶ DataSource

#### ▶ DefaultFields

- ▶ Designer
- ▶ Eof
- ▶ FieldCount
- ▶ FieldDefList
- ▶ FieldDefs
- ▶ FieldList
- ▶ Fields

#### FieldValues

#### ▶ Found

- ▶ Modified
- ▶ Name
- ▶ ObjectView
- ▶ SparseArrays
- ▶ State

### Derived from TComponent

#### ▶ ComObject

- ▶ ComponentCount
- ▶ ComponentIndex
- ▶ Components
- ▶ ComponentState

▶ ComponentStyle  
▶ DesignInfo  
▶ Owner  
▶ Tag  
▶ VCLComObject

## TIBQuery properties

[TIBQuery](#)

[By object](#)

[Legend](#)

▶	<u><a href="#">Active</a></u>
▶	▶ <u><a href="#">AggFields</a></u>
▶	<u><a href="#">AutoCalcFields</a></u>
▶	▶ <u><a href="#">Bof</a></u>
	<u><a href="#">Bookmark</a></u>
▶	<u><a href="#">CachedUpdates</a></u>
▶	▶ <u><a href="#">ComObject</a></u>
▶	▶ <u><a href="#">ComponentCount</a></u>
	<u><a href="#">ComponentIndex</a></u>
▶	<u><a href="#">Components</a></u>
▶	▶ <u><a href="#">ComponentState</a></u>
▶	▶ <u><a href="#">ComponentStyle</a></u>
▶	<u><a href="#">Constraints</a></u>
▶	<u><a href="#">Database</a></u>
	<u><a href="#">DatasetField</a></u>
▶	▶ <u><a href="#">DataSource</a></u>
▶	<u><a href="#">DBHandle</a></u>
▶	▶ <u><a href="#">DefaultFields</a></u>
▶	▶ <u><a href="#">Designer</a></u>
	<u><a href="#">DesignInfo</a></u>
▶	▶ <u><a href="#">Eof</a></u>
▶	▶ <u><a href="#">FieldCount</a></u>
▶	▶ <u><a href="#">FieldDefList</a></u>
	<u><a href="#">FieldDefs</a></u>
▶	<u><a href="#">FieldList</a></u>
▶	▶ <u><a href="#">Fields</a></u>
	<u><a href="#">FieldValues</a></u>
▶	▶ <u><a href="#">Found</a></u>
	<u><a href="#">GenerateParamNames</a></u>
▶	▶ <u><a href="#">Modified</a></u>
▶	<u><a href="#">Name</a></u>
▶	<u><a href="#">ObjectView</a></u>
▶	<u><a href="#">Owner</a></u>
▶	<u><a href="#">ParamCheck</a></u>
▶	▶ <u><a href="#">ParamCount</a></u>
▶	<u><a href="#">Params</a></u>
	<u><a href="#">Prepared</a></u>
▶	▶ <u><a href="#">RowsAffected</a></u>
▶	▶ <u><a href="#">SparseArrays</a></u>
▶	<u><a href="#">SQL</a></u>
▶	▶ <u><a href="#">State</a></u>
▶	▶ <u><a href="#">StmtHandle</a></u>
▶	<u><a href="#">Tag</a></u>
▶	<u><a href="#">Text</a></u>
▶	<u><a href="#">TRHandle</a></u>
▶	<u><a href="#">Transaction</a></u>
▶	<u><a href="#">UniDirectional</a></u>
▶	<u><a href="#">UpdateObject</a></u>
▶	<u><a href="#">UpdateRecordTypes</a></u>
▶	▶ <u><a href="#">UpdatesPending</a></u>
	<u><a href="#">VCLComObject</a></u>

## TIBQuery.GenerateParamNames

[TIBQuery](#)    [See also](#)

Generates a list of parameter names for the query.

**property** GenerateParamNames: Boolean;

### Description

Set GenerateParamNames to True to have the query generate a list of parameter names.

## TIBQuery.ParamCheck

[TIBQuery](#)   [See also](#)

Specifies whether the parameter list for a query is regenerated if the SQL property changes at runtime.

**property** ParamCheck: Boolean;

### Description

Set ParamCheck to specify whether or not the Params property is cleared and regenerated if an application modifies the query's SQL property at runtime. By default ParamCheck is True, meaning that the Params property is automatically regenerated at runtime. When ParamCheck is True, the proper number of parameters is guaranteed to be generated for the current SQL statement.

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



## TIBQuery.ParamCount

[TIBQuery](#)   [See also](#)   [Example](#)

Indicates the current number of parameters for the query.

**property** ParamCount: Word;

### Description

Inspect ParamCount to determine how many parameters are in the Params property. If the ParamCheck property is True, ParamCount always corresponds to the number of actual parameters in the SQL statement for the query.

## TIBQuery.Params

[TIBQuery](#)   [See also](#)   [Example](#)

Contains the parameters for a query's SQL statement.

**property** Params: TParams;

### Description

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

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

## TIBQuery.Prepared

[TIBQuery](#)

[See also](#)

[Example](#)

Determines whether or not a query is prepared for execution.

**property** Prepared: Boolean;

### Description

Examine Prepared to determine if a query is already prepared for execution. If Prepared is True, the query is prepared, and if Prepared is False, the query is not prepared. While a query need not be prepared before execution, execution performance is enhanced if the query is prepared beforehand, particularly if it is a parameterized query that is executed more than once using the same parameter values.

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

## TIBQuery.RowsAffected

[TIBQuery](#)    [See also](#)

Returns the number of rows operated upon by the latest query execution.

**property** RowsAffected: Integer;

### Description

Check RowsAffected to determine how many rows were updated or deleted by the last query operation. If RowsAffected is -1, the query did not update or delete any rows.

## TIBQuery.SQL

[TIBQuery](#)

[See also](#)

[Example](#)

Contains the text of the SQL statement to execute for the query.

**property** SQL: TStrings;

### Description

Use SQL to provide the SQL statement that a query component executes when its ExecSQL or Open method is called. At design time the SQL property can be edited by invoking the String List editor in the Object Inspector.

The SQL property may contain only one complete SQL statement at a time.

## TIBQuery.StmtHandle

### TIBQuery

Identifies the statement handle for the query.

**property** StmtHandle: TISC\_STMT\_HANDLE;

### **Description**

Retrieve StmtHandle if an application makes a direct call to the InterBase server, bypassing the methods of TIBQuery. Some API calls require a statement handle as a parameter. Under all other circumstances an application does not need to access this property.

## TIBQuery.Text

[TIBQuery](#)    [See also](#)

Points to the actual text of the SQL query.

**property** Text: **String**;

### Description

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

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

## TIBQuery.UniDirectional

### TIBQuery

Determines whether or not bidirectional cursors are enabled for a query's result set.

**property** UniDirectional: Boolean;

### **Description**

Set UniDirectional to control whether or not a cursor can move forward and backward through a result set. By default UniDirectional is False, enabling forward and backward navigation.

**Note:** If an application does not need bidirectional access to records in a result set, set UniDirectional to True. When UniDirectional is True, an application requires less memory and performance is improved.



## TIBQuery.UpdateObject

### TIBQuery

Specifies the update object component used to update a read-only result set when cached updates are enabled.

**property** UpdateObject: TIBDataSetUpdateObject;

### **Description**

Set UpdateObject to specify the update object component used to update a read-only result set when cached updates are enabled.

## TIBQuery events

[TIBQuery](#)

[Alphabetically Legend](#)

### Derived from TIBCustomDataSet

- ▶ [OnUpdateError](#)
- ▶ [OnUpdateRecord](#)

### Derived from TDataSet

- ▶ [AfterCancel](#)
- ▶ [AfterClose](#)
- ▶ [AfterDelete](#)
- ▶ [AfterEdit](#)
- ▶ [AfterInsert](#)
- ▶ [AfterOpen](#)
- ▶ [AfterPost](#)
- ▶ [AfterRefresh](#)
- ▶ [AfterScroll](#)
- ▶ [BeforeCancel](#)
- ▶ [BeforeClose](#)
- ▶ [BeforeDelete](#)
- ▶ [BeforeEdit](#)
- ▶ [BeforeInsert](#)
- ▶ [BeforeOpen](#)
- ▶ [BeforePost](#)
- ▶ [BeforeRefresh](#)
- ▶ [BeforeScroll](#)
- ▶ [OnCalcFields](#)
- ▶ [OnDeleteError](#)
- ▶ [OnEditError](#)
- ▶ [OnFilterRecord](#)
- ▶ [OnNewRecord](#)
- ▶ [OnPostError](#)

## TIBQuery events

[TIBQuery](#)   [By object](#)   [Legend](#)

▶	<a href="#">AfterCancel</a>
▶	<a href="#">AfterClose</a>
▶	<a href="#">AfterDelete</a>
▶	<a href="#">AfterEdit</a>
▶	<a href="#">AfterInsert</a>
▶	<a href="#">AfterOpen</a>
▶	<a href="#">AfterPost</a>
▶	<a href="#">AfterRefresh</a>
▶	<a href="#">AfterScroll</a>
▶	<a href="#">BeforeCancel</a>
▶	<a href="#">BeforeClose</a>
▶	<a href="#">BeforeDelete</a>
▶	<a href="#">BeforeEdit</a>
▶	<a href="#">BeforeInsert</a>
▶	<a href="#">BeforeOpen</a>
▶	<a href="#">BeforePost</a>
▶	<a href="#">BeforeRefresh</a>
▶	<a href="#">BeforeScroll</a>
▶	<a href="#">OnCalcFields</a>
▶	<a href="#">OnDeleteError</a>
▶	<a href="#">OnEditError</a>
▶	<a href="#">OnFilterRecord</a>
▶	<a href="#">OnNewRecord</a>
▶	<a href="#">OnPostError</a>
▶	<a href="#">OnUpdateError</a>
▶	<a href="#">OnUpdateRecord</a>

## TIBQuery methods

[TIBQuery](#)    [Alphabetically](#)

### In TIBQuery

[Create](#)  
[Destroy](#)  
[ExecSQL](#)  
[GetDetailLinkFields](#)  
[ParamByName](#)  
[Prepare](#)  
[UnPrepare](#)

### Derived from TIBCustomDataSet

[ApplyUpdates](#)  
[CachedUpdateStatus](#)  
[CancelUpdates](#)  
[CreateBlobStream](#)  
[BatchInput](#)  
[BatchOutput](#)  
[FetchAll](#)  
[GetCurrentRecord](#)  
[GetFieldData](#)  
[Locate](#)  
[LocateNext](#)  
[Lookup](#)  
[RecordModified](#)  
[RevertRecord](#)  
[Undelete](#)  
[UpdateStatus](#)

### Derived from TDataSet

[ActiveBuffer](#)  
[Append](#)  
[AppendRecord](#)  
[CheckBrowseMode](#)  
[ClearFields](#)  
[Close](#)  
[CompareBookmarks](#)  
[ControlsDisabled](#)  
[CursorPosChanged](#)  
[Delete](#)  
[DisableControls](#)  
[Edit](#)  
[EnableControls](#)  
[FieldByName](#)  
[FindField](#)  
[FindFirst](#)

FindLast  
FindNext  
FindPrior  
First  
FreeBookmark  
GetBookmark  
GetDetailDataSets  
GetDetailLinkFields  
GetFieldList  
GetFieldNames  
GetProviderAttributes  
GotoBookmark  
Insert  
InsertRecord  
IsEmpty  
IsLinkedTo  
Last  
MoveBy  
Next  
Open  
Post  
Prior  
Refresh  
Resync  
SetFields  
Translate  
UpdateCursorPos  
UpdateRecord

**Derived from TComponent**

DestroyComponents  
Destroying  
ExecuteAction  
FindComponent  
FreeNotification  
FreeOnRelease  
GetNamePath  
GetParentComponent  
HasParent  
InsertComponent  
RemoveComponent  
SafeCallException  
UpdateAction

**Derived from TPersistent**

Assign

**Derived from TObject**AfterConstructionBeforeDestructionClassInfoClassNameClassNamelsClassParentClassTypeCleanupInstanceDefaultHandlerDispatchFieldAddressFreeFreeInstanceGetInterfaceGetInterfaceEntryGetInterfaceTableInheritsFromInitInstanceInstanceSizeMethodAddressMethodNameNewInstance

## TIBQuery methods

[TIBQuery](#)   [By object](#)

[ActiveBuffer](#)  
[AfterConstruction](#)  
[Append](#)  
[AppendRecord](#)  
[ApplyUpdates](#)  
[Assign](#)  
[BeforeDestruction](#)  
[CachedUpdateStatus](#)  
[CancelUpdates](#)  
[CheckBrowseMode](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[ClearFields](#)  
[Close](#)  
[CompareBookmarks](#)  
[ControlsDisabled](#)  
[Create](#)  
[CreateBlobStream](#)  
[CursorPosChanged](#)  
[DefaultHandler](#)  
[Delete](#)  
[Destroy](#)  
[DestroyComponents](#)  
[Destroying](#)  
[DisableControls](#)  
[Dispatch](#)  
[Edit](#)  
[EnableControls](#)  
[ExecSQL](#)  
[ExecuteAction](#)  
[BatchInput](#)  
[BatchOutput](#)  
[FetchAll](#)  
[FieldAddress](#)  
[FieldByName](#)  
[FindComponent](#)  
[FindField](#)  
[FindFirst](#)

FindLast  
FindNext  
FindPrior  
First  
Free  
FreeBookmark  
FreeInstance  
FreeNotification  
FreeOnRelease  
GetBookmark  
GetCurrentRecord  
GetDetailDataSets  
GetDetailLinkFields  
GetFieldData  
GetFieldList  
GetFieldNames  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
GetNamePath  
GetParentComponent  
GetProviderAttributes  
GotoBookmark  
HasParent  
InheritsFrom  
InitInstance  
Insert  
InsertComponent  
InsertRecord  
InstanceSize  
IsEmpty  
IsLinkedTo  
Last  
Locate  
LocateNext  
Lookup  
MethodAddress  
MethodName  
MoveBy  
NewInstance  
Next  
Open  
ParamByName  
Post  
Prepare



Prior

RecordModified

Refresh

RemoveComponent

Resync

RevertRecord

SafeCallException

SetFields

Translate

Undelete

UnPrepare

UpdateAction

UpdateCursorPos

UpdateRecord

UpdateStatus

## TIBQuery.Create

[TIBQuery](#)    [See also](#)

Creates an instance of a query component.

**constructor** Create (AOwner: TComponent) ;

### Description

Call Create to instantiate a query at runtime. Query components placed in forms or data modules at design time are created automatically.

Create calls its inherited Create constructor, creates an empty SQL statement list, creates an empty parameter list, sets the OnChange event handler for the SQL statement list, sets the ParamCheck property to True, and sets the RowsAffected property to -1.

## TIBQuery.Destroy

[TIBQuery](#)    [See also](#)

Destroys the instance of a query.

**destructor** Destroy;

### Description

Do not call Destroy directly. Instead call Free to verify that the query is not already freed before calling Destroy. Destroy disconnects from the server, frees the SQL statement list and the parameter list, and then calls its inherited destructor.

## TIBQuery.ExecSQL

[TIBQuery](#)

[See also](#)

[Example](#)

Executes the SQL statement for the query.

**procedure** ExecSQL;

### Description

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

**Note:** For SELECT statements, call Open instead of ExecSQL.

ExecSQL prepares the statement in SQL property for execution if it has not already been prepared. To speed performance, an application should ordinarily call Prepare before calling ExecSQL for the first time.

## **TIBQuery.GetDetailLinkFields**

[TIBCustomDataSet](#)

[See also](#)

Fills lists with the master and detail fields of the link.

**procedure** GetDetailLinkFields (MasterFields, DetailFields: TList);

### **Description**

Creates two lists of TFields from the master-detail relationship between two tables; one containing the master fields, and the other containing the detail fields.

## TIBQuery.ParamByName

[TIBQuery](#)

[See also](#)

[Example](#)

Accesses parameter information based on a specified parameter name.

**function** ParamByName(**const** Value: **string**): TParam;

### Description

Call ParamByName to set or use parameter information for a specific parameter based on its name.

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

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

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

## TIBQuery.Prepare

[TIBQuery](#)

[See also](#)

[Example](#)

Sends a query to the server for optimization prior to execution.

**procedure** Prepare;

### Description

Call Prepare to have the remote database server allocate resources for the query and to perform additional optimizations. Calling Prepare before executing a query improves application performance.

Delphi automatically prepares a query if it is executed without first being prepared. After execution, Delphi unprepares the query. When a query will be executed a number of times, an application should always explicitly prepare the query to avoid multiple and unnecessary prepares and unprepares.

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

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

## TIBQuery.UnPrepare

[TIBQuery](#)    [See also](#)

Frees the resources allocated for a previously prepared query.

**procedure** UnPrepare;

### Description

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

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

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



## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



TDataSet



TIBCustomDataSet

## TIBSQL

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBSQL provides an object for executing an InterBase SQL statement with minimal overhead.

### Unit

IBSQL

### Description

Use a TIBSQL object to execute an InterBase SQL statement with minimal overhead. TIBSQL has no standard interface to data-aware controls and is unidirectional.

## TIBSQL properties

[TIBSQL](#)

[Alphabetically Legend](#)

### In TIBSQL

- ▶ [Bof](#)
- ▶ [Database](#)
- ▶ [DBHandle](#)
- ▶ [Eof](#)
- ▶ [FieldIndex](#)
- ▶ [Fields](#)
- ▶ [GenerateParamNames](#)
- ▶ [GoToFirstRecordOnExecute](#)
- ▶ [Handle](#)
- ▶ [Open](#)
- ▶ [ParamCheck](#)
- ▶ [Params](#)
- ▶ [Plan](#)
- ▶ [Prepared](#)
- ▶ [RecordCount](#)
- ▶ [RowsAffected](#)
- ▶ [SQL](#)
- ▶ [SQLType](#)
- ▶ [Transaction](#)
- ▶ [TRHandle](#)
- ▶ [UniqueRelationName](#)

### Derived from TComponent

- ▶ [ComObject](#)
- ▶ [ComponentCount](#)
- ▶ [ComponentIndex](#)
- ▶ [Components](#)
- ▶ [ComponentState](#)
- ▶ [ComponentStyle](#)
- ▶ [DesignInfo](#)
- ▶ [Name](#)
- ▶ [Owner](#)
- ▶ [Tag](#)
- ▶ [VCLComObject](#)

## TIBSQL properties

[TIBSQL](#)

[By object](#)

[Legend](#)

### ► [Bof](#)

- [ComObject](#)
- [ComponentCount](#)
- [ComponentIndex](#)
- [Components](#)
- [ComponentState](#)
- [ComponentStyle](#)

### [DesignInfo](#)

- [Database](#)
- [DBHandle](#)
- [Eof](#)
- [FieldIndex](#)
- [Fields](#)
- [GenerateParamNames](#)
- [GoToFirstRecordOnExecute](#)
- [Handle](#)
- [Name](#)
- [Open](#)
- [Owner](#)
- [ParamCheck](#)
- [Params](#)
- [Plan](#)
- [Prepared](#)
- [RecordCount](#)
- [RowsAffected](#)
- [SQL](#)
- [SQLType](#)
- [Tag](#)
- [Transaction](#)
- [TRHandle](#)
- [UniqueRelationName](#)
- [VCLComObject](#)

## TIBSQL.Bof

[TIBSQL](#)      [See also](#)

Indicates whether or not a query is at the beginning of the dataset.

**property** Bof: Boolean;

Use the Bof property to determine whether or not a query is at the beginning of the dataset.

## **TIBSQL.Database**

[TIBSQL](#)

[See also](#)

Sets or returns the database associated with the query.

**property** Database: [TIBDatabase](#);

### **Description**

Use the Database property to set or return the database associated with the query.



## TIBSQL.DBHandle

[TIBSQL](#)

[See also](#)

Specifies the database handle for the query.

**property** DBHandle: PISC\_DB\_HANDLE;

### Description

Use the DBHandle property to return the database handle for the query.

## TIBSQL.Eof

[TIBSQL](#)      [See also](#)

Indicates whether or not a query is at the end of the dataset.

**property** Eof: Boolean;

### Description

Use the Eof property to determine whether or not a query is at the end of the dataset and whether or not the query returned any result rows.

In addition, if this property is True immediately after the query is opened, then it means that the dataset is empty.

## TIBSQL.FieldIndex

[TIBSQL](#)

[See also](#)

Returns the index of the named field.

**property** FieldIndex: [FieldName: **String**]: Integer;

### Description

Use the FieldIndex property to return the index of the name field.

## TIBSQL.Fields

[TIBSQL](#)      [See also](#)

Returns the XSQLVAR fields.

**property** Fields[const Idx: Integer]: TIBXSQLVAR;

### Description

Use the Fields property to return the XSQLVAR fields.

## **TIBSQL.GenerateParamNames**

[TIBSQL](#)

[See also](#)

Generates a list of parameter names for the query.

**property** GenerateParamNames: Boolean;

### **Description**

Set GenerateParamNames to True to have the query generate a list of parameter names.

## TIBSQL.GoToFirstRecordOnExecute

[TIBSQL](#)

[See also](#)

Goes to the first record in the result set upon opening it.

**property** GoToFirstRecordOnExecute: Boolean;

### Description

Use the GoToFirstRecordOnExecute property to go to the first record in a result set upon opening it. By default, this property is set to True. GoToFirstRecordOnExecute exists primarily for use in TIBDataSet, which sets this value to False for its internal TIBSQLs.

## TIBSQL.Handle

[TIBSQL](#)

[See also](#)

Specifies the handle for the query.

**property** Handle: TISC\_STMT\_HANDLE;

### Description

Use the Handle property to get the query handle.

## TIBSQL.Open

[TIBSQL](#)

[See also](#)

Determines if the dataset is open.

**property** Open: Boolean;

### Description

Use the Open property to determine if the dataset is open.



## TIBSQL.ParamCheck

[TIBSQL](#)

[See also](#)

Specifies whether the parameter list for an SQL query is regenerated if the SQL property changes at runtime.

**property** ParamCheck: Boolean;

### Description

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

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

## TIBSQL.Params

[TIBSQL](#)      [See also](#)

Returns the XSQLDA parameters.

**property** Params: TIBXSQLDA;

### Description

Use the Params property to return the XSQLDA parameters.

## TIBSQL.Plan

[TIBSQL](#)      [See also](#)

Returns the plan for the query.

**property** Plan: **String**;

### Description

Use the Plan property to view the query plan once the query has been prepared.

## TIBSQL.Prepared

[TIBSQL](#)

[See also](#)

[Example](#)

Indicates whether or not the query has been prepared.

**property** Prepared: Boolean;

### Description

Use the Prepared property to determine whether or not a query has yet been prepared.

## TIBSQL.RecordCount

[TIBSQL](#)

[See also](#)

Returns the current count of records from the query.

**property** RecordCount: Integer;

### Description

Use the RecordCount property to see how many records are returned by a query. If the result set is to return 100 rows, RecordCount will only be 100 after all the records have been visited. That is, after looking at the first record, RecordCount is 1, and so forth.

## TIBSQL.RowsAffected

[TIBSQL](#)

[See also](#)

Returns the number of rows affected.

**property** RowsAffected: Integer;

### Description

Use the RowsAffected property to return the number of rows affected by the query. This property is useful for INSERT, DELETE, and UPDATE statements.

## TIBSQL.SQL

[TIBSQL](#)      [See also](#)

Sets the SQL query to be executed.

**property** SQL: TString;

### Description

Use the SQL property to write or view the SQL query to be executed.

## TIBSQL.SQLType

[TIBSQL](#)      [See also](#)

Returns the type of query to be executed.

```
type TIBSQLTypes = set of (SQLUnknown, SQLSelect, SQLInsert, SQLUpdate,  
    SQLDelete, SQLDDL, SQLGetSegment, SQLPutSegment, SQLExecProcedure,  
    SQLStartTransaction, SQLCommit, SQLRollback, SQLSelectForUpdate,  
    SQLSetGenerator);
```

```
property SQLType: TIBSQLTypes read FSQLType;
```

### Description

Use the SQLType to determine the type of query to be executed. Query types include:

SQLCommit	Commits an active transaction
SQLDDL	Modifies the database metadata
SQLDelete	Removes rows in a table or in the active set of a cursor
SQLExecProcedure	Calls a stored procedure
SQLGetSegment	Reads a segment from an open Blob
SQLInsert	Adds one or more new rows to a specified table
SQLPutSegment	Writes a Blob segment
SQLRollback	Restores the database to its state prior to the start of the current transaction
SQLSelectForUpdate	Used for positioned updates.
SQLSetGenerator	Sets a new value for an existing generator
SQLSelect	Retrieves data from one or more tables
SQLStartTransaction	Starts a new transaction against one or more databases
SQLUnknown	Unknown SQL type
SQLUpdate	Changes data in all or part of an existing row in a table, view, or active set of a cursor

## TIBSQL.Transaction

[TIBSQL](#)      [See also](#)

Sets or returns the transaction to be used by the query.

```
property Transaction: TIBTransaction;
```

### Description

Use the Transaction property to set or return the transaction to be used by the query.



## **TIBSQL.TRHandle**

[TIBSQL](#)

[See also](#)

Specifies the transaction handle for the query.

**property** TRHandle: PISC\_TR\_HANDLE;

### **Description**

Use the TRHandle property to return the transaction handle for the query.

## TIBSQL.UniqueRelationName

[TIBSQL](#)

[See also](#)

Indicates the unique relation name.

**property** UniqueRelationName: String;

### Description

Use the UniqueRelationName property to indicate the unique relation name for a query that involves only one base table.

## TIBDynSQL events

[TIBSQL](#)

[Alphabetically Legend](#)


In TIBSQL



[OnSQLChanging](#)

TIBDynSQL events

[TIBSQL](#)   [By object](#)   [Legend](#)

 OnSQLChanging

## TIBSQL.OnSQLChanging

[TIBSQL](#)

[See also](#)

Occurs when the SQL query is being modified.

**property** OnSQLChanging: TNotifyEvent;

### Description

Write an OnSQLChanging event handler to take specific actions when a query is being modified. If an exception is raised in this event, the query is not changed.

## TIBSQL methods

[TIBSQL](#)      [Alphabetically](#)

### In TIBSQL

[BatchInput](#)

[BatchOutput](#)

[Call](#)

[CheckClosed](#)

[CheckOpen](#)

[CheckValidStatement](#)

[Close](#)

[Create](#)

[Current](#)

[Destroy](#)

[ExecQuery](#)

[FieldByName](#)

[FreeHandle](#)

[Next](#)

[Prepare](#)

### Derived from TComponent

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### Derived from TPersistent

[Assign](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNames](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

Dispatch  
FieldAddress  
Free  
FreeInstance  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
InheritsFrom  
InitInstance  
InstanceSize  
MethodAddress  
MethodName  
NewInstance

## TIBSQL methods

[TIBSQL](#)

[By object](#)

[AfterConstruction](#)

[Assign](#)

[BatchInput](#)

[BatchOutput](#)

[BeforeDestruction](#)

[Call](#)

[CheckClosed](#)

[CheckOpen](#)

[CheckValidStatement](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Close](#)

[Create](#)

[Current](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecQuery](#)

[ExecuteAction](#)

[FieldAddress](#)

[FieldByName](#)

[FindComponent](#)

[Free](#)

[FreeInstance](#)

[FreeHandle](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)



InstanceSize

MethodAddress

MethodName

NewInstance

Next

Prepare

RemoveComponent

SafeCallException

UpdateAction

## **TIBSQL.BatchInput**

[TIBSQL](#)

[See also](#)

Executes the parameterized query in SQL for input in the referenced input object.

**procedure** BatchInput (InputObject: [TIBBatchInput](#));

### **Description**

Call BatchInput to execute the parameterized query in SQL for input in the referenced input object.

## TIBSQL.BatchOutput

[TIBSQL](#)

[See also](#)

Outputs the selected query in SQL to the referenced OutputObject.

**procedure** BatchOutput (OutputObject: TIBBatchOutput);

### Description

Call BatchOutput to output the selected query in SQL to the referenced OutputObject.

## TIBSQL.Call

[TIBSQL](#)      [See also](#)

Returns an error message based on the error code.

**function** Call(ErrCode: ISC\_STATUS; RaiseError: Boolean): ISC\_STATUS;

### Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning an error based on the value of RaiseError.

## **TIBSQL.CheckClosed**

[TIBSQL](#)

[See also](#)

Raises an exception if the query is not closed.

**procedure** CheckClosed;

### **Description**

Call CheckClosed to raise an exception if the query is not closed.

## **TIBSQL.CheckOpen**

[TIBSQL](#)

[See also](#)

Raises an exception if the query is not open.

**procedure** CheckOpen;

### **Description**

Call CheckOpen to raise an exception if the query is closed.

## **TIBSQL.CheckValidStatement**

[TIBSQL](#)      [See also](#)

Raises an exception if the query does not have a valid statement.

**procedure** CheckValidStatement;

### **Description**

Call CheckValidStatement to raise an exception if the query does not have a valid statement.

## **TIBSQL.Close**

[TIBSQL](#)      [See also](#)

Closes the query.

**procedure** Close;

### **Description**

Call Close to close the query.



## TIBSQL.Create

[TIBSQL](#)

[See also](#)

Creates an instance of a TIBSQL component.

**constructor** `Create (AOwner:TComponent) ;`

### Description

Call Create to create an instance of a TIBSQL component.

## TIBSQL.Current

[TIBSQL](#)      [See also](#)

Returns an extended SQL descriptor for the current record.

**function** Current: TIBXSQlda;

### Description

Call Current to get an extended SQL descriptor for the current record.

## TIBSQL.Destroy

[TIBSQL](#)

[See also](#)

Frees all resources associated with this instance.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Usually destruction of objects is handled automatically by Delphi. If an application creates its own instance of an update object, however, the application should call Free, which verifies that the update object is not already freed before calling Destroy.

## **TIBSQL.ExecQuery**

[TIBSQL](#)      [See also](#)

Executes an SQL query.

**procedure** ExecQuery;

### **Description**

Call ExecQuery to execute the SQL query.

## TIBSQL.FieldByName

[TIBSQL](#)

[See also](#)

Returns the XSQLVAR fields by name.

**function** FieldByName[FieldName: String]: TIBXSQLVAR;

### Description

Use the FieldByName method to return the XSQLVAR fields by name.

## **TIBSQL.FreeHandle**

[TIBSQL](#)

[See also](#)

Frees InterBase resources associated with the query.

**procedure** FreeHandle;

### **Description**

Call FreeHandle to free the InterBase resources associated with the query.

## TIBSQL.Next

[TIBSQL](#)      [See also](#)

Returns an extended SQL descriptor for the next record.

**function** Next: TIBXSQlda;

### Description

Call Next to get an extended SQL descriptor for the next record.

## TIBSQL.Prepare

[TIBSQL](#)

[See also](#)

[Example](#)

Prepares a query for execution.

**procedure** Prepare;

### Description

Call Prepare to prepare a query for execution.



## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent

## **TIBCustomSQLMonitor**

[Hierarchy](#)   [Properties](#)   [Methods](#)   [See also](#)

TIBCustomSQLMonitor is the ancestor object from which TIBSQLMonitor and TIBSQLMonitorHook are derived.

### **Unit**

IBSQLMonitor

### **Description**

TIBCustomSQLMonitor is the ancestor object from which TIBSQLMonitor and TIBSQLMonitorHook are derived.

## TIBCustomSQLMonitor properties

[TIBCustomSQLMonitor](#)

[Alphabetically](#) [Legend](#)

### Derived from TComponent

▶ <a href="#">ComObject</a>	
▶	<a href="#">ComponentCount</a>
	<a href="#">ComponentIndex</a>
▶	<a href="#">Components</a>
▶	<a href="#">ComponentState</a>
▶	<a href="#">ComponentStyle</a>
	<a href="#">DesignInfo</a>
▶	<a href="#">Name</a>
	▶ <a href="#">Owner</a>
▶	<a href="#">Tag</a>
	<a href="#">VCLComObject</a>

## TIBCustomSQLMonitor properties

[TIBCustomSQLMonitor](#)

[By object](#)

[Legend](#)

### ▶ ComObject

- ▶ ComponentCount
- ▶ ComponentIndex
- ▶ Components
- ▶ ComponentState
- ▶ ComponentStyle
- ▶ DesignInfo
- ▶ Name
- ▶ Owner
- ▶ Tag
- ▶ VCLComObject

## TIBCustomSQLMonitor methods

[TIBCustomSQLMonitor](#)

[Alphabetically](#)

### In TIBCustomSQLMonitor

[Create](#)

[Destroy](#)

### Derived from TComponent

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### Derived from TPersistent

[Assign](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNames](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

## TIBCustomSQLMonitor methods

[TIBCustomSQLMonitor](#)

[By object](#)

[AfterConstruction](#)

[Assign](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecuteAction](#)

[FieldAddress](#)

[FindComponent](#)

[Free](#)

[FreeInstance](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)



## TIBCustomSQLMonitor.Create

[TIBCustomSQLMonitor](#)      [See also](#)

Creates an instance of TIBCustomSQLMonitor.

**constructor** Create (AOwner: TComponent);

### Description

Call Create to create an instance of TIBCustomSQLMonitor. Create also:

- Creates the window
- Registers the event
- Fires the Monitor thread

## **TIBCustomSQLMonitor.Destroy**

[TIBCustomSQLMonitor](#)

[See also](#)

Destroys the instance of TIBCustomSQLMonitor.

**destructor** Destroy;

### **Description**

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy.

Destroy unregisters the event, destroys the window, and kills the Monitor thread.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



## TIBSQLMonitor

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBSQLMonitor monitors dynamic SQL passed to the InterBase server.

### Unit

IBSQLMonitor

### Description

Use TIBSQLMonitor to monitor dynamic SQL taking place in InterBase applications. Enable TraceFlags in each TIBDatabase component in order for the SQL monitor to receive status information from each database connection.

## TIBSQLMonitor properties

[TIBSQLMonitor](#)

[Alphabetically](#) [Legend](#)

### Derived from TComponent

▶ <a href="#">ComObject</a>	
▶	<a href="#">ComponentCount</a>
	<a href="#">ComponentIndex</a>
▶	<a href="#">Components</a>
▶	<a href="#">ComponentState</a>
▶	<a href="#">ComponentStyle</a>
	<a href="#">DesignInfo</a>
▶	<a href="#">Name</a>
	▶ <a href="#">Owner</a>
▶	<a href="#">Tag</a>
	<a href="#">VCLComObject</a>

TIBSQLMonitor properties

[TIBSQLMonitor](#)

[By object](#)

[Legend](#)

▶ <a href="#">ComObject</a>	
▶	<a href="#">ComponentCount</a>
	<a href="#">ComponentIndex</a>
▶	<a href="#">Components</a>
▶	<a href="#">ComponentState</a>
▶	<a href="#">ComponentStyle</a>
	<a href="#">DesignInfo</a>
▶	<a href="#">Name</a>
	▶ <a href="#">Owner</a>
▶	<a href="#">Tag</a>
	<a href="#">VCLComObject</a>



## TIBSQLMonitor events

[TIBSQLMonitor](#)

[Alphabetically](#) [Legend](#)

In TIBSQLMonitor



OnSQL

TIBSQLMonitor events

[TIBSQLMonitor](#)

[By object](#)

[Legend](#)



OnSQL

## TIBSQLMonitor.OnSQL

[TIBSQLMonitor](#)

[See also](#)

Reports dynamic SQL activity on InterBase applications.

**property** OnSQL: TSQLEvent;

### Description

Write an OnSQL event handler to report dynamic SQL activity on InterBase applications. OnSQL is an event of type TSQLEvent, and reports SQL activity through the EventText:

TSQLEvent = procedure(EventText: String) of object;

You must enable the TraceFlags in each TIBDatabase component in order for the SQL monitor to receive status information from each database connection.

## **TIBSQLMonitor methods**

[TIBSQLMonitor](#)

[Alphabetically](#)

### **Derived from TIBCustomSQLMonitor**

[Create](#)

[Destroy](#)

### **Derived from TComponent**

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### **Derived from TPersistent**

[Assign](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNames](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

## TIBSQLMonitor methods

[TIBSQLMonitor](#)

[By object](#)

[AfterConstruction](#)

[Assign](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecuteAction](#)

[FieldAddress](#)

[FindComponent](#)

[Free](#)

[FreeInstance](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



TIBCustomSQLMonitor



## **TIBSQLMonitorHook**

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBSQLMonitorHook is an internal object used by the components to output messages for use by TIBSQLMonitor.

### **Unit**

IBSQLMonitor

### **Description**

TIBSQLMonitorHook is an internal object used by the components to output messages for use by TIBSQLMonitor.

## TIBSQLMonitorHook properties

[TIBSQLMonitorHook](#)

[Alphabetically](#) [Legend](#)

### In TIBSQLMonitorHook

[TraceFlags](#)

### Derived from TComponent

▸ [ComObject](#)

▸ [ComponentCount](#)

[ComponentIndex](#)

▸ [Components](#)

▸ [ComponentState](#)

▸ [ComponentStyle](#)

[DesignInfo](#)

▸ [Name](#)

▸ [Owner](#)

▸ [Tag](#)

[VCLComObject](#)

TIBSQLMonitorHook properties

[TIBSQLMonitorHook](#)      [By object](#)      [Legend](#)

- ▶ [ComObject](#)
  - ▶ [ComponentCount](#)
  - ▶ [ComponentIndex](#)
  - ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
  - ▶ [DesignInfo](#)
  - ▶ [Name](#)
  - ▶ [Owner](#)
  - ▶ [Tag](#)
  - ▶ [TraceFlags](#)
  - ▶ [VCLComObject](#)

## TIBSQLMonitorHook.TraceFlags

[TIBSQLMonitorHook](#)

[See also](#)

This is an internal property used by the TIBDatabase component.

**property** TraceFlags: TTraceFlags;

### Description

Use TraceFlags to specify which database operations the SQL Monitor should track in an application at runtime. TraceFlags is only meaningful for the SQL Monitor, which is provided to enable performance tuning and SQL debugging when working with remote SQL database servers.

**Note:** Normally trace options are set from the SQL Monitor rather than setting TraceFlags in application code.

## TIBSQLMonitorHook events

[TIBSQLMonitorHook](#)

[Alphabetically](#) [Legend](#)

Derived from TIBSQLMonitor



[OnSQL](#)

TIBSQLMonitorHook events

[TIBSQLMonitorHook](#)

[By object](#)

[Legend](#)



OnSQL

## TIBSQLMonitorHook methods

[TIBSQLMonitorHook](#)

[Alphabetically](#)

### In TIBSQLMonitorHook

[Create](#)

[DBConnect](#)

[DBDisconnect](#)

[Destroy](#)

[MonitorCount](#)

[ReadSQLData](#)

[RegisterMonitor](#)

[ServiceAttach](#)

[ServiceDetach](#)

[ServiceQuery](#)

[ServiceStart](#)

[SQLExecute](#)

[SQLFetch](#)

[SQLPrepare](#)

[TRCommit](#)

[TRCommitRetaining](#)

[TRRollback](#)

[TRRollbackRetaining](#)

[TRStart](#)

[UnregisterMonitor](#)

### Derived from TComponent

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### Derived from TPersistent

[Assign](#)

### Derived from TObject

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

ClassNamels  
ClassParent  
ClassType  
CleanupInstance  
DefaultHandler  
Dispatch  
FieldAddress  
Free  
FreeInstance  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
InheritsFrom  
InitInstance  
InstanceSize  
MethodAddress  
MethodName  
NewInstance



## TIBSQLMonitorHook methods

[TIBSQLMonitorHook](#)

[By object](#)

[AfterConstruction](#)

[Assign](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[DBConnect](#)

[DBDisconnect](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecuteAction](#)

[FieldAddress](#)

[FindComponent](#)

[Free](#)

[FreeInstance](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[MonitorCount](#)

[NewInstance](#)

[ReadSQLData](#)

[RegisterMonitor](#)

[RemoveComponent](#)

[SafeCallException](#)

ServiceAttach  
ServiceDetach  
ServiceQuery  
ServiceStart  
SQLExecute  
SQLFetch  
SQLPrepare  
TRCommit  
TRCommitRetaining  
TRRollback  
TRRollbackRetaining  
TRStart  
UnregisterMonitor  
UpdateAction

## TIBSQLMonitorHook.Create

[TIBSQLMonitorHook](#)

[See also](#)

Creates an instance of TIBSQLMonitorHook.

**constructor** Create;

### Description

Call Create to create an instance of TIBSQLMonitorHook. Create also:

- Creates a mapped memory file
- Creates the event
- Sets the initial state

## TIBSQLMonitorHook.DBCConnect

[TIBSQLMonitorHook](#)

[See also](#)

Outputs database connect notifications.

**procedure** DBCConnect (db: TIBDatabase) ;

### Description

Call the DBCConnect method to output database connect notifications.

## TIBSQLMonitorHook.DBDisconnect

[TIBSQLMonitorHook](#)

[See also](#)

Outputs database disconnect notifications.

**procedure** DBDisconnect (db: TIBDatabase) ;

### Description

Call the DBdisconnect method to output database disconnect notifications.

## TIBSQLMonitorHook.Destroy

[TIBSQLMonitorHook](#)

[See also](#)

Destroys the instance of TIBSQLMonitorHook.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead, an application should call Free. Free verifies that the service object has not already been freed before it calls Destroy. Destroy unmaps the memory-mapped file and closes the event handles.

## **TIBSQLMonitorHook.MonitorCount**

[TIBSQLMonitorHook](#)

[See also](#)

Returns the number of monitors.

**function** MonitorCount: Integer;

### **Description**

Call MonitorCount to get the number of active monitors.

## TIBSQLMonitorHook.ReadSQLData

[TIBSQLMonitorHook](#)

[See also](#)

Returns the contents of the notification buffer for the enabled trace flags.

**function** ReadSQLData: **String**;

### Description

Call ReadSQLData to return the contents of the notification buffer for appropriately enabled trace flags.



## TIBSQLMonitorHook.RegisterMonitor

[TIBSQLMonitorHook](#)

[See also](#)

Registers monitors.

**procedure** RegisterMonitor;

### Description

Call RegisterMonitor to register monitors.

## TIBSQLMonitorHook.ServiceAttach

[TIBSQLMonitorHook](#)

[See also](#)

Outputs a service attach notification.

**procedure** ServiceAttach(qry: TIBCustomService);

### Description

Call ServiceAttach to output a service attach notification.

## TIBSQLMonitorHook.ServiceDetach

[TIBSQLMonitorHook](#)

[See also](#)

Outputs a service detach notification.

**procedure** ServiceDetach(qry: TIBCustomService);

### Description

Call ServiceDetach to output a service detach notification.

## TIBSQLMonitorHook.ServiceQuery

[TIBSQLMonitorHook](#)

[See also](#)

Outputs a service query notification.

**procedure** ServiceQuery(qry: TIBCustomService);

### Description

Call ServiceQuery to output a service query notification.

## TIBSQLMonitorHook.ServiceStart

[TIBSQLMonitorHook](#)

[See also](#)

Outputs a service start notification.

**procedure** ServiceStart (qry: TIBCustomService);

### Description

Call ServiceStart to output a service start notification.

## **TIBSQLMonitorHook.SQLExecute**

[TIBSQLMonitorHook](#)

[See also](#)

Outputs the SQL execute notification.

**procedure** SQLExecute (qry: TIBSQL) ;

### **Description**

Call SQLExecute to output the SQL execute notification, along with the query text and parameters.

## TIBSQLMonitorHook.SQLFetch

[TIBSQLMonitorHook](#)

[See also](#)

Outputs the SQL fetch notification.

**procedure** SQLFetch (qry: TIBSQL) ;

### Description

Call SQLFetch to output the SQL fetch notification, along with the SQL statement and the status of the fetch (for example, whether EOF has been reached, etc).

## **TIBSQLMonitorHook.SQLPrepare**

[TIBSQLMonitorHook](#)

[See also](#)

Outputs the SQL prepare notification.

**procedure** SQLPrepare (qry: TIBSQL) ;

### **Description**

Call SQLPrepare to output the SQL prepare notification, along with the query, text, and plan.



## **TIBSQLMonitorHook.TRCommit**

[TIBSQLMonitorHook](#)

[See also](#)

Outputs the commit notification.

**procedure** TRCommit (tr: [TIBTransaction](#));

### **Description**

Call TRCommit to output the commit notification.

## **TIBSQLMonitorHook.TRCommitRetaining**

[TIBSQLMonitorHook](#)

[See also](#)

Outputs the commit retaining notification.

**procedure** TRCommitRetaining(tr: TIBTransaction);

### **Description**

Call TRCommitRetaining to output the commit retaining notification.

## TIBSQLMonitorHook.TRRollback

[TIBSQLMonitorHook](#)

[See also](#)

Outputs the rollback notification.

**procedure** TRRollback(tr: TIBTransaction);

### Description

Call TRRollback to output the rollback notification.

## **TIBSQLMonitorHook.TRRollbackRetaining**

[TIBSQLMonitorHook](#)

[See also](#)

Outputs the rollback retaining notification.

**procedure** TRRollbackRetaining(tr: [TIBTransaction](#));

### **Description**

Call TRRollbackRetaining to output the rollback retaining notification.

## **TIBSQLMonitorHook.TRStart**

[TIBSQLMonitorHook](#)

[See also](#)

Outputs the start transaction notification.

**procedure** TRStart(tr: [TIBTransaction](#));

### **Description**

Call TRStart to output the transaction start notification.

## **TIBSQLMonitorHook.UnregisterMonitor**

[TIBSQLMonitorHook](#)

[See also](#)

Unregisters the monitor.

**procedure** UnregisterMonitor;

### **Description**

Call UnregisterMonitor to unregister the monitor.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published



## Hierarchy

TObject



TPersistent



TComponent



TIBCustomSQLMonitor



TIBSQLMonitor



## TIBStoredProc

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBStoredProc encapsulates a stored procedure on a database server.

### Unit

IBStoredProc

### Description

Use a TIBStoredProc object when a client application must use a stored procedure on a database server. A stored procedure is a grouped set of statements, stored as part of a database server's metadata (just like tables, indexes, and domains), that performs a frequently repeated, database-related task on the server and passes results to the client.

Many stored procedures require a series of input arguments, or parameters, that are used during processing. TIBStoredProc provides a Params property that enables an application to set these parameters before executing the stored procedure.

TIBStoredProc reuses the Params property to hold the results returned by a stored procedure. Params is an array of values. A stored procedure will return a single set of values, or none at all.

**Note:** TIBStoredProc components should be used with InterBase Execute stored procedures only. To use InterBase Select procedures, use TIBQuery or TIBDataSet. Since Execute stored procedures do not return result sets, never use Open or Active on the TIBStoredProc component. Use ExecProc instead.

## TIBStoredProc properties

[TIBStoredProc](#) [Alphabetically](#) [Legend](#)

### In TIBStoredProc

- ▶ [NameList](#)
  - ▶ [ParamCount](#)
  - ▶ [Params](#)
  - ▶ [Prepared](#)
  - ▶ [StmtHandle](#)
  - ▶ [StoredProcName](#)

### Derived from TCustomDataSet

- ▶ [CachedUpdates](#)
- ▶ [Database](#)
- ▶ [DBHandle](#)
- ▶ [Transaction](#)
- ▶ [TRHandle](#)
- ▶ [UpdateObject](#)
- ▶ [UpdateRecordTypes](#)
- ▶ [UpdatesPending](#)

### Derived from TDataSet

- ▶ [Active](#)
- ▶ [AggFields](#)
- ▶ [AutoCalcFields](#)
- ▶ [Bof](#)
- ▶ [Bookmark](#)
- ▶ [DatasetField](#)
- ▶ [DataSource](#)
- ▶ [DefaultFields](#)
- ▶ [Designer](#)
  - ▶ [Eof](#)
  - ▶ [FieldCount](#)
  - ▶ [FieldDefList](#)
  - ▶ [FieldDefs](#)
  - ▶ [FieldList](#)
  - ▶ [Fields](#)
  - ▶ [FieldValues](#)
- ▶ [Found](#)
  - ▶ [Modified](#)
  - ▶ [Name](#)
  - ▶ [ObjectView](#)
  - ▶ [SparseArrays](#)
  - ▶ [State](#)

### Derived from TComponent

- ▶ [ComObject](#)
  - ▶ [ComponentCount](#)
  - ▶ [ComponentIndex](#)
  - ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
  - ▶ [DesignInfo](#)
- ▶ [Owner](#)
  - ▶ [Tag](#)
  - ▶ [VCLComObject](#)

## TIBStoredProc properties

[TIBStoredProc](#) [By object](#)

[Legend](#)

- ▶ [Active](#)
- ▶ ▶ [AggFields](#)
- ▶ [AutoCalcFields](#)
- ▶ ▶ [Bof](#)
- ▶ [Bookmark](#)
- ▶ [CachedUpdates](#)
- ▶ ▶ [ComObject](#)
- ▶ ▶ [ComponentCount](#)
- ▶ [ComponentIndex](#)
- ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
- ▶ [Database](#)
- ▶ ▶ [DataSource](#)
- ▶ [DBHandle](#)
- ▶ ▶ [DefaultFields](#)
- ▶ [Designer](#)
  - ▶ [DesignInfo](#)
  - ▶ [Eof](#)
  - ▶ [FieldCount](#)
  - ▶ [FieldDefList](#)
  - ▶ [FieldDefs](#)
  - ▶ [FieldList](#)
  - ▶ [Fields](#)
- ▶ [FieldValues](#)
- ▶ [Found](#)
  - ▶ [Modified](#)
- ▶ ▶ [Name](#)
- ▶ ▶ [NameList](#)
- ▶ ▶ [ObjectView](#)
- ▶ ▶ [Owner](#)
- ▶ ▶ [ParamCount](#)
- ▶ ▶ [Params](#)
- ▶ [Prepared](#)
- ▶ [SparseArrays](#)
  - ▶ [State](#)
  - ▶ [StmtHandle](#)
- ▶ [StoredProcName](#)
- ▶ [Tag](#)
- ▶ [TRHandle](#)
- ▶ [Transaction](#)
- ▶ [UpdateObject](#)
- ▶ [UpdateRecordTypes](#)
- ▶ ▶ [UpdatesPending](#)
- ▶ [VCLComObject](#)

## **TIBStoredProc.NameList**

[TIBStoredProc](#) [See also](#)

Returns a list of stored procedures in the database.

**property** NameList: TStrings;

### **Description**

NameList is an internal property used to list the stored procedures available in the database.

## TIBStoredProc.ParamCount

[TIBStoredProc](#) [See also](#) [Example](#)

Indicates the number of parameters for the stored procedure component.

**property** ParamCount: Word;

### Description

Examine ParamCount to determine the number of parameters currently stored in the Params property.

## TIBStoredProc.Params

[TIBStoredProc](#) [See also](#) [Example](#)

Stores the input and output parameters for a stored procedure.

**property** Params: TParams;

### Description

Access Params at runtime to set input parameter names, values, and data types dynamically (at design time use the Parameters editor to set parameter information). Params is an array of parameter values.

An application can also access Params after executing a stored procedure to retrieve the output parameters returned to the procedure by the server.

## TIBStoredProc.Prepared

[TIBStoredProc](#) [See also](#)

Determines whether or not a stored procedure is prepared for execution.

**property** Prepared: Boolean;

### Description

Examine Prepared to determine if a stored procedure is already prepared for execution. If Prepared is True, the stored procedure is prepared, and if Prepared is False, the procedure is not prepared. A stored procedure must be prepared before it can be executed.

**Note:** Delphi automatically prepares a stored procedure if it is unprepared when the application calls ExecProc. After execution, Delphi automatically unprepares the stored procedure. If a procedure will be executed a number of times, it is more efficient for the application to prepare the stored procedure once, and unprepare it when it is no longer needed.

An application can change the current setting of Prepared to prepare or unprepare a stored procedure. If Prepared is True, setting it to False calls the Unprepare method to unprepare the stored procedure. If Prepared is False, setting it to True calls the Prepare method to prepare the procedure. Generally, however, it is better programming practice to call Prepare and Unprepare directly. These methods automatically update the Prepared property.



## TIBStoredProc.StmtHandle

### [TIBStoredProc](#)

Identifies the statement handle for the stored procedure.

**property** StmtHandle: TISC\_STMT\_HANDLE;

#### Description

Retrieve StmtHandle if an application makes a direct call to the InterBase server, bypassing the methods of TIBStoredProc. Some API calls require a statement handle as a parameter. Under all other circumstances an application does not need to access this property.

## **TIBStoredProc.StoredProcName**

[TIBStoredProc](#) [See also](#)

Identifies the name of the stored procedure on the server for which this object is an encapsulation.

**property** StoredProcName: **String**;

### **Description**

Set StoredProcName to specify the name of the stored procedure to call on the server. If StoredProcName does not match the name of an existing stored procedure on the server, then when the application attempts to prepare the procedure prior to execution, an exception is raised.

## TIBStoredProc events

[TIBStoredProc](#) [Alphabetically](#) [Legend](#)

### Derived from TIBCustomDataSet

- ▶ [OnUpdateError](#)
- ▶ [OnUpdateRecord](#)

### Derived from TDataSet

- ▶ [AfterCancel](#)
- ▶ [AfterClose](#)
- ▶ [AfterDelete](#)
- ▶ [AfterEdit](#)
- ▶ [AfterInsert](#)
- ▶ [AfterOpen](#)
- ▶ [AfterPost](#)
- ▶ [AfterRefresh](#)
- ▶ [AfterScroll](#)
- ▶ [BeforeCancel](#)
- ▶ [BeforeClose](#)
- ▶ [BeforeDelete](#)
- ▶ [BeforeEdit](#)
- ▶ [BeforeInsert](#)
- ▶ [BeforeOpen](#)
- ▶ [BeforePost](#)
- ▶ [BeforeRefresh](#)
- ▶ [BeforeScroll](#)
- ▶ [OnCalcFields](#)
- ▶ [OnDeleteError](#)
- ▶ [OnEditError](#)
- ▶ [OnNewRecord](#)
- ▶ [OnPostError](#)

## TIBStoredProc events

[TIBStoredProc](#) [By object](#) [Legend](#)

▶	<a href="#">AfterCancel</a>
▶	<a href="#">AfterClose</a>
▶	<a href="#">AfterDelete</a>
▶	<a href="#">AfterEdit</a>
▶	<a href="#">AfterInsert</a>
▶	<a href="#">AfterOpen</a>
▶	<a href="#">AfterPost</a>
▶	<a href="#">AfterRefresh</a>
▶	<a href="#">AfterScroll</a>
▶	<a href="#">BeforeCancel</a>
▶	<a href="#">BeforeClose</a>
▶	<a href="#">BeforeDelete</a>
▶	<a href="#">BeforeEdit</a>
▶	<a href="#">BeforeInsert</a>
▶	<a href="#">BeforeOpen</a>
▶	<a href="#">BeforePost</a>
▶	<a href="#">BeforeRefresh</a>
▶	<a href="#">BeforeScroll</a>
▶	<a href="#">OnCalcFields</a>
▶	<a href="#">OnDeleteError</a>
▶	<a href="#">OnEditError</a>
▶	<a href="#">OnNewRecord</a>
▶	<a href="#">OnPostError</a>
▶	<a href="#">OnUpdateError</a>
▶	<a href="#">OnUpdateRecord</a>

## **TIBStoredProc methods**

[TIBStoredProc Alphabetically](#)

### **In TIBStoredProc**

[CopyParams](#)

[Create](#)

[Destroy](#)

[ExecProc](#)

[ParamByName](#)

[Prepare](#)

[UnPrepare](#)

### **Derived from TIBCustomDataSet**

[ApplyUpdates](#)

[BatchInput](#)

[BatchOutput](#)

[CachedUpdateStatus](#)

[CancelUpdates](#)

[CreateBlobStream](#)

[FetchAll](#)

[GetCurrentRecord](#)

[GetFieldData](#)

[Locate](#)

[LocateNext](#)

[Lookup](#)

[RecordModified](#)

[RevertRecord](#)

[Undelete](#)

[UpdateStatus](#)

### **Derived from TDataSet**

[ActiveBuffer](#)

[Append](#)

[AppendRecord](#)

[CheckBrowseMode](#)

[ClearFields](#)

[Close](#)

[CompareBookmarks](#)

[ControlsDisabled](#)

[CursorPosChanged](#)

[Delete](#)

[DisableControls](#)

[Edit](#)

[EnableControls](#)

[FieldByName](#)

[FindField](#)

[FindFirst](#)

[FindLast](#)  
[FindNext](#)  
[FindPrior](#)  
[First](#)  
[FreeBookmark](#)  
[GetBookmark](#)  
[GetDetailDataSets](#)  
[GetDetailLinkFields](#)  
[GetFieldList](#)  
[GetFieldNames](#)  
[GetProviderAttributes](#)  
[GotoBookmark](#)  
[Insert](#)  
[InsertRecord](#)  
[IsEmpty](#)  
[IsLinkedTo](#)  
[Last](#)  
[MoveBy](#)  
[Next](#)  
[Open](#)  
[Post](#)  
[Prior](#)  
[Refresh](#)  
[Resync](#)  
[SetFields](#)  
[Translate](#)  
[UpdateCursorPos](#)  
[UpdateRecord](#)

**Derived from TComponent**

[DestroyComponents](#)  
[Destroying](#)  
[ExecuteAction](#)  
[FindComponent](#)  
[FreeNotification](#)  
[FreeOnRelease](#)  
[GetNamePath](#)  
[GetParentComponent](#)  
[HasParent](#)  
[InsertComponent](#)  
[RemoveComponent](#)  
[SafeCallException](#)  
[UpdateAction](#)

**Derived from TPersistent**

[Assign](#)

**Derived from TObject**

AfterConstruction

BeforeDestruction

ClassInfo

ClassName

ClassNamels

ClassParent

ClassType

CleanupInstance

DefaultHandler

Dispatch

FieldAddress

Free

FreeInstance

GetInterface

GetInterfaceEntry

GetInterfaceTable

InheritsFrom

InitInstance

InstanceSize

MethodAddress

MethodName

NewInstance

## TIBStoredProc methods

[TIBStoredProc By object](#)

[ActiveBuffer](#)  
[AfterConstruction](#)  
[Append](#)  
[AppendRecord](#)  
[ApplyUpdates](#)  
[Assign](#)  
[BatchInput](#)  
[BatchOutput](#)  
[BeforeDestruction](#)  
[CachedUpdateStatus](#)  
[CancelUpdates](#)  
[CheckBrowseMode](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[ClearFields](#)  
[Close](#)  
[CompareBookmarks](#)  
[ControlsDisabled](#)  
[CopyParams](#)  
[Create](#)  
[CreateBlobStream](#)  
[CursorPosChanged](#)  
[DefaultHandler](#)  
[Delete](#)  
[Destroy](#)  
[DestroyComponents](#)  
[Destroying](#)  
[DisableControls](#)  
[Dispatch](#)  
[Edit](#)  
[EnableControls](#)  
[ExecProc](#)  
[ExecuteAction](#)  
[FetchAll](#)  
[FieldAddress](#)  
[FieldByName](#)  
[FindComponent](#)  
[FindField](#)



FindFirst  
FindLast  
FindNext  
FindPrior  
First  
Free  
FreeBookmark  
FreeInstance  
FreeNotification  
FreeOnRelease  
GetBookmark  
GetCurrentRecord  
GetDetailDataSets  
GetDetailLinkFields  
GetFieldData  
GetFieldList  
GetFieldNames  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
GetNamePath  
GetParentComponent  
GetProviderAttributes  
GotoBookmark  
HasParent  
InheritsFrom  
InitInstance  
Insert  
InsertComponent  
InsertRecord  
InstanceSize  
IsEmpty  
IsLinkedTo  
Last  
Locate  
LocateNext  
Lookup  
MethodAddress  
MethodName  
MoveBy  
NewInstance  
Next  
Open  
ParamByName  
Post

Prepare

Prior

RecordModified

Refresh

RemoveComponent

Resync

RevertRecord

SafeCallException

SetFields

Translate

UpdateCursorPos

UpdateRecord

Undelete

UnPrepare

UpdateAction

UpdateStatus

## TIBStoredProc.CopyParams

[TIBStoredProc](#) [See also](#)

Copies a stored procedure's parameters into another parameter list.

**procedure** CopyParams (Value: TParams) ;

### Description

Call CopyParams to copy this stored procedure's parameters into a separate parameter list object. Value is the parameter list into which to assign this stored procedure's parameters. Value can be the parameter list of another stored procedure. For example:

```
TIBStoredProc1.CopyParams (TIBStoredProc2.Params) ;
```

If the stored procedure is not prepared when an application calls CopyParams, CopyParams calls Prepare before assigning the parameters to the target parameters list, and then calls UnPrepare to return the stored procedure to its previous state.

## TIBStoredProc.Create

[TIBStoredProc](#) [See also](#)

Creates an instance of a stored procedure component.

**constructor** Create (AOwner: TComponent) ;

### Description

Call Create to instantiate a stored procedure declared in an application. Create calls its inherited Create constructor, creates an empty parameter list for the newly instantiated stored procedure, and initializes its parameter, server, and record buffers to nil.

## **TIBStoredProc.Destroy**

[TIBStoredProc](#) [See also](#)

Destroys the instance of a stored procedure.

**destructor** Destroy;

### **Description**

Do not call Destroy directly. Instead call Free to verify that the stored procedure is not already freed before calling Destroy. Destroy disconnects from the server, frees the parameter list, and calls its inherited Destroy destructor.

## TIBStoredProc.ExecProc

[TIBStoredProc](#) [See also](#) [Example](#)

Executes the stored procedure on the server.

**procedure** ExecProc;

### Description

Call ExecProc to execute a stored procedure on the server. Before calling ExecProc:

- 1 Provide any input parameters in the Params property. At design time, a developer can provide parameters using the Parameters editor. At runtime an application must access Params directly.
- 2 Call Prepare to bind the parameters.

If a stored procedure returns output parameters, they are stored in the Params property when ExecProc returns control to the application. An application can access the output parameters by indexing into the Params list, or by using the ParamByName method.

## TIBStoredProc.ParamByName

[TIBStoredProc](#) [See also](#) [Example](#)

Accesses parameter information based on a specified parameter name.

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

### Description

Call ParamByName to return parameter information for a specific parameter based on its name. Value is the name of the parameter for which to retrieve information. Typically ParamByName is used to set an input parameter's value at runtime, or to retrieve the value of an output parameter. The following command line assigns the value "Jane Smith" as the value for the parameter named Contact:

```
StoredProc1.ParamByName('Contact').AsString := 'Jane Smith';
```

## TIBStoredProc.Prepare

[TIBStoredProc](#) [See also](#) [Example](#)

Prepares a stored procedure for execution.

**procedure** Prepare;

### Description

Call Prepare to bind a stored procedure's parameters before calling ExecProc to execute the procedure. Prepare readies a stored procedure's parameters and informs the server of the stored procedure's readiness. These steps allocate system resources and optimize the query for server performance.

**Note:** If an application attempts to execute a stored procedure that has not been prepared, Delphi automatically prepares the procedure before executing it, and then unprepares it when execution is complete. If a stored procedure will be executed more than once, it is more efficient for an application to call Prepare explicitly once to avoid repeated and unnecessary preparing and unpreparing of the stored procedure, and then call UnPrepare when the stored procedure is no longer needed.



## **TIBStoredProc.UnPrepare**

[TIBStoredProc](#) [See also](#)

Frees the resources allocated for a previously prepared stored procedure.

**procedure** UnPrepare;

### **Description**

Call UnPrepare to free the resources allocated for a previously prepared stored procedure on the server and client sides.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



TDataSet



TIBCustomDataSet

## TIBStringField

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBStringField allows for strings in excess of 8196 bytes of data.

### Unit

IBCustomDataSet

### Description

A value of a string field is physically stored as a sequence of characters. Common uses for string fields are to store text, such as names and addresses.

TIBStringField introduces properties to translate between string values and other data types, and to manage language driver conversions. As a descendent of TStringField, TIBStringField inherits many properties, methods, and events that are useful for managing the value and properties of a field in a database. TIBStringField allows for strings in excess of 8196 bytes of data.

## TIBStringField properties

[TIBStringField](#) [Alphabetically](#) [Legend](#)

### In TIBStringField

- ▶ [BlanksToNULL](#)

### Derived from TStringField

- [AsBoolean](#)
- [AsDateTime](#)
- [AsFloat](#)
- [AsInteger](#)
- [AsString](#)
- [AsVariant](#)
- ▶ [DataSize](#)
- ▶ [FixedChar](#)
- ▶ [Transliterate](#)
- [Value](#)

### Derived from TField

- ▶ [Alignment](#)
- [AsBoolean](#)
- [AsDateTime](#)
- [AttributeSet](#)
- ▶ [AutoGenerateValue](#)
- [Calculated](#)
- ▶ [CanModify](#)
- ▶ [ConstraintErrorMessage](#)
- ▶ [CurValue](#)
- ▶ [CustomConstraint](#)
- [DataSet](#)
- ▶ [DataType](#)
- ▶ [DefaultExpression](#)
- ▶ [DisplayLabel](#)
- ▶ [DisplayName](#)
- ▶ [DisplayText](#)
- ▶ [DisplayWidth](#)
- [EditMask](#)
- ▶ [EditMaskPtr](#)
- ▶ [FieldKind](#)
- ▶ [FieldName](#)
- ▶ [FieldNo](#)
- ▶ [FullName](#)
- ▶ [HasConstraints](#)
- ▶ [ImportedConstraint](#)
- ▶ [Index](#)
- ▶ [IsIndexField](#)
- ▶ [IsNull](#)
- ▶ [KeyFields](#)
- [Lookup](#)
- ▶ [LookupCache](#)
- ▶ [LookupDataSet](#)
- ▶ [LookupKeyFields](#)
- ▶ [LookupList](#)
- ▶ [LookupResultField](#)

▶▶



**►**



**P**



### Derived

**Derived from TComponent**

ComObject

ComponentCount

## ComponentIndex

## Components

## ComponentState

## ComponentStyle

## DesignInfo

Name

OwnerTagVCLComObject

## TIBStringField properties

[TIBStringField](#) [By object](#) [Legend](#)

▶	<u>Alignment</u>
	<u>AsBoolean</u>
	<u>AsDateTime</u>
	<u>AsFloat</u>
	<u>AsInteger</u>
	<u>AsString</u>
	<u>AsVariant</u>
	<u>AttributeSet</u>
▶	<u>AutoGenerateValue</u>
▶	<u>BlanksToNULL</u>
	<u>Calculated</u>
▶	<u>CanModify</u>
▶	<u>ComObject</u>
▶	<u>ComponentCount</u>
	<u>ComponentIndex</u>
▶	<u>Components</u>
▶	<u>ComponentState</u>
▶	<u>ComponentStyle</u>
▶	<u>ConstraintErrorMessage</u>
▶	<u>CurValue</u>
▶	<u>CustomConstraint</u>
	<u>DataSet</u>
▶	<u>DataSetSize</u>
▶	<u>DataType</u>
▶	<u>DefaultExpression</u>
	<u>DesignInfo</u>
▶	<u>DisplayLabel</u>
▶	<u>DisplayName</u>
▶	<u>DisplayText</u>
▶	<u>DisplayWidth</u>
	<u>EditMask</u>
▶	<u>EditMaskPtr</u>
▶	<u>FieldKind</u>
▶	<u>FieldName</u>
▶	<u>FieldNo</u>
▶	<u>FixedChar</u>
▶	<u>FullName</u>
▶	<u>HasConstraints</u>
▶	<u>ImportedConstraint</u>
▶	<u>Index</u>
▶	<u>IsIndexField</u>
▶	<u>IsNull</u>
▶	<u>KeyFields</u>
	<u>Lookup</u>
▶	<u>LookupCache</u>
▶	<u>LookupDataSet</u>
▶	<u>LookupKeyFields</u>
▶	<u>LookupList</u>
▶	<u>LookupResultField</u>
▶	<u>Name</u>
	<u>NewValue</u>



▶ Offset  
▶ OldValue  
▶ Origin  
▶ Owner  
  
▶ ParentField  
▶ ProviderFlags  
▶ ReadOnly  
▶ Required  
▶ Tag  
▶ Text  
▶ Transliterate  
  
ValidChars  
Value  
VCLComObject  
▶ Visible

## TIBStringField.BlanksToNULL

[TIBStringField](#) [See also](#)

Converts blank spaces to null.

**property** BlanksToNull: Boolean;

### Description

Use BlanksToNULL to convert blank spaces to null within a dataset.

## **TIBStringField methods**

[TIBStringField](#) [Alphabetically](#)

### **In TIBStringField**

[CheckTypeSize](#)

[Create](#)

[GetAsString](#)

[GetAsVariant](#)

[GetValue](#)

[SetAsString](#)

### **Derived from TField**

[Assign](#)

[AssignValue](#)

[Clear](#)

[Destroy](#)

[FocusControl](#)

[GetData](#)

[IsBlob](#)

[IsValidChar](#)

[RefreshLookupList](#)

[SetData](#)

[SetFieldType](#)

[Validate](#)

### **Derived from TComponent**

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### **Derived from TPersistent**

[Assign](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

ClassType  
CleanupInstance  
DefaultHandler  
Dispatch  
FieldAddress  
Free  
FreeInstance  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
InheritsFrom  
InitInstance  
InstanceSize  
MethodAddress  
MethodName  
NewInstance

## TIBStringField methods

[TIBStringField](#) [By object](#)

[AfterConstruction](#)

[Assign](#)

[AssignValue](#)

[BeforeDestruction](#)

[CheckTypeSize](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Clear](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecuteAction](#)

[FieldAddress](#)

[FindComponent](#)

[FocusControl](#)

[Free](#)

[FreeInstance](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetAsString](#)

[GetAsVariant](#)

[GetData](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetParentComponent](#)

[GetValue](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)

[InstanceSize](#)

[IsBlob](#)

[IsValidChar](#)

[MethodAddress](#)

MethodName

NewInstance

RefreshLookupList

RemoveComponent

SafeCallException

SetAsString

SetData

SetFieldType

UpdateAction

Validate

## **TIBStringField.CheckTypeSize**

[TIBStringField](#) [See also](#)

Indicates the type size of the string field.

**class procedure** CheckTypeSize (Value: Integer);

### **Description**

Call CheckTypeSize to determine the type size of the string field.

All sizes are valid for a TIBString field.

## TIBStringField.Create

[TIBStringField](#) [See also](#)

Creates an instance of a TIBStringField object.

**constructor** `Create (AOwner: TComponent);`

### Description

It is seldom necessary to call Create directly, because a string field component is instantiated automatically for all string fields in a dataset.

After calling the inherited constructor, Create sets

- DataType to ftString.
- Size to 20.
- Transliterate to True.
- BlanksToNull to true



## **TIBStringField.GetAsString**

[TIBStringField](#) [See also](#)

Returns the value of a field as a string.

**function** GetAsString: string;

### **Description**

Call GetAsString to return the value of a field as a string.

## **TIBStringField.GetAsVariant**

[TIBStringField](#) [See also](#)

Returns the value of a field as a variant.

**function** GetAsVariant: Variant;

### **Description**

Call GetAsVariant return the value of a field as type Variant.

## TIBStringField.GetValue

[TIBStringField](#) [See also](#)

Returns the value of a field.

**function** GetValue(var Value: string): Boolean;

### Description

Call GetValue to return the value of a field.

## TIBStringField.SetAsString

[TIBStringField](#) [See also](#)

Sets the value of a field as a string type.

**procedure** SetAsString(const Value: string);

### Description

Call SetAsString to set the value of a field as a string type.

## TIBStringField events

[TIBStringField](#) [Alphabetically](#) [Legend](#)

### Derived from TField

- ▶ [OnChange](#)
- ▶ [OnGetText](#)
- ▶ [OnSetText](#)
- ▶ [OnValidate](#)

## TIBStringField events

[TIBStringField](#) [By object](#) [Legend](#)

▢	<u>OnChange</u>
▢	<u>OnGetText</u>
▢	<u>OnSetText</u>
▢	<u>OnValidate</u>

## Scope



Published

## Accessibility



Read-only

**Scope**



Published



## Hierarchy

TObject



TPersistent



TComponent



TField



TStringField



## TIBTable

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBTable is a dataset component that encapsulates a database table.

### Unit

IBTable

### Description

Use TIBTable to access data in a single table or view. TIBTable provides direct access to every record and field in an underlying InterBase database table. A table component can also work with a subset of records within a database table using filters.

## TIBTable properties

[TIBTable](#)

[Alphabetically Legend](#)

### In TIBTable

- ▶ [BufferChunks](#)
- ▶ ▶ [CurrentDBKey](#)
- ▶ ▶ [DefaultIndex](#)
- ▶ ▶ [Exists](#)
- ▶ ▶ [Filter](#)
- ▶ ▶ [Filtered](#)
- ▶ ▶ [IndexDefs](#)
- ▶ ▶ [IndexFieldCount](#)
- ▶ ▶ [IndexFieldNames](#)
- ▶ ▶ [IndexFields](#)
- ▶ ▶ [IndexName](#)
- ▶ ▶ [MasterFields](#)
- ▶ ▶ [MasterSource](#)
- ▶ ▶ [ReadOnly](#)
- ▶ ▶ [StoreDefs](#)
- ▶ ▶ [TableName](#)
- ▶ ▶ [TableNames](#)
- ▶ ▶ [TableTypes](#)
- ▶ ▶ [UniDirectional](#)
- ▶ ▶ [UpdateObject](#)

### Derived from TIBCustomDataSet

- ▶ ▶ [CachedUpdates](#)
- ▶ ▶ [Database](#)
- ▶ ▶ [DBHandle](#)
- ▶ ▶ [Transaction](#)
- ▶ ▶ [TRHandle](#)
- ▶ ▶ [UpdateRecordTypes](#)
- ▶ ▶ [UpdatesPending](#)

### Derived from TDataSet

- ▶ ▶ [Active](#)
- ▶ ▶ [AggFields](#)
- ▶ ▶ [AutoCalcFields](#)
- ▶ ▶ [BlockReadSize](#)
- ▶ ▶ [Bof](#)
- ▶ ▶ [Bookmark](#)
- ▶ ▶ [Constraints](#)
- ▶ ▶ [DataSetField](#)
- ▶ ▶ [DefaultFields](#)
  - ▶ ▶ [Designer](#)
  - ▶ ▶ [Eof](#)
  - ▶ ▶ [FieldCount](#)
  - ▶ ▶ [FieldDefList](#)
- ▶ ▶ [FieldDefs](#)
- ▶ ▶ [FieldList](#)
- ▶ ▶ [Fields](#)
  - ▶ ▶ [FieldValues](#)
- ▶ ▶ [FilterOptions](#)
- ▶ ▶ [Found](#)
- ▶ ▶ [Modified](#)
- ▶ ▶ [Name](#)

- ▶ ObjectView  
RecNo
- ▶ RecordCount
  - ▶ RecordSize
  - ▶ SparseArrays
  - ▶ State

**Derived from TComponent**

- ▶ ComObject
  - ▶ ComponentCount
  - ▶ ComponentIndex
  - ▶ Components
  - ▶ ComponentState
  - ▶ ComponentStyle
  - ▶ DesignInfo
- ▶ Owner
- ▶ Tag  
VCLComObject

## TIBTable properties

[TIBTable](#)

[By object](#)

[Legend](#)

▶	<a href="#">Active</a>
▶	▶ <a href="#">AggFields</a>
▶	<a href="#">AutoCalcFields</a>
	<a href="#">BlockReadSize</a>
	▶ <a href="#">Bof</a>
	<a href="#">Bookmark</a>
▶	<a href="#">BufferChunks</a>
▶	<a href="#">CachedUpdates</a>
	▶ <a href="#">ComObject</a>
▶	<a href="#">ComponentCount</a>
	<a href="#">ComponentIndex</a>
	▶ <a href="#">Components</a>
	▶ <a href="#">ComponentState</a>
	▶ <a href="#">ComponentStyle</a>
▶	<a href="#">Constraints</a>
	▶ <a href="#">CurrentDBKey</a>
▶	<a href="#">Database</a>
	<a href="#">DataSetField</a>
	▶ <a href="#">DBHandle</a>
	▶ <a href="#">DefaultFields</a>
▶	<a href="#">DefaultIndex</a>
	▶ <a href="#">Designer</a>
	<a href="#">DesignInfo</a>
	▶ <a href="#">Eof</a>
	▶ <a href="#">Exists</a>
	▶ <a href="#">FieldCount</a>
▶	<a href="#">FieldDefList</a>
▶	<a href="#">FieldDefs</a>
	▶ <a href="#">FieldList</a>
	▶ <a href="#">Fields</a>
	<a href="#">FieldValues</a>
▶	<a href="#">Filter</a>
▶	<a href="#">Filtered</a>
▶	<a href="#">Filter</a>
▶	<a href="#">FilterOptions</a>
▶	<a href="#">Found</a>
▶	<a href="#">IndexDefs</a>
	▶ <a href="#">IndexFieldCount</a>
▶	<a href="#">IndexFieldNames</a>
	<a href="#">IndexFields</a>
▶	<a href="#">IndexName</a>
▶	<a href="#">MasterFields</a>
▶	<a href="#">MasterSource</a>
	▶ <a href="#">Modified</a>
▶	<a href="#">Name</a>
▶	<a href="#">ObjectView</a>
▶	<a href="#">Owner</a>
▶	<a href="#">ReadOnly</a>
	<a href="#">RecNo</a>
	▶ <a href="#">RecordCount</a>
	▶ <a href="#">RecordSize</a>

- [SparseArrays](#)
    - [State](#)
  - [StoreDefs](#)
  - [TableName](#)
  - [TableNames](#)
  - [TableTypes](#)
  - [Tag](#)
  - [Transaction](#)
  - [TRHandle](#)
  - [UniDirectional](#)
  - [UpdateObject](#)
  - [UpdateRecordTypes](#)
  - [UpdatesPending](#)
    - [VCLComObject](#)

## **TIBTable.BufferChunks**

[TIBTable](#)      [See also](#)

Indicates the buffer incrementation size.

**property** BufferChunks: Integer;

### **Description**

Use the BufferChunks property to indicate the chunk size (in records) used to increment the buffer.

## **TIBTable.CurrentDBKey**

[TIBTable](#)      [See also](#)

Returns the DBKey for the current row in the table.

**property** CurrentDBKey: TIBDBKey;

### **Description**

Use CurrentDBKey to return the DBKey for the current row in the table. A DBKey is a unique row identifier for the duration of the current transaction. This property is primarily for internal use.



## TIBTable.DefaultIndex

[TIBTable](#)

[See also](#)

Specifies if the data in the table should be ordered on a default index when opened.

**property** DefaultIndex: Boolean;

### Description

When this property is set to False, an ORDER BY clause is not used when opening a table on the server. When DefaultIndex is True, the data is ordered based on the primary key or a unique index when opening the table. DefaultIndex defaults to True.

## TIBTable.Exists

[TIBTable](#)

[See also](#)

[Example](#)

Indicates whether the underlying database table exists.

**property** Exists: Boolean;

### Description

Read Exists at runtime to determine whether a database table exists. If the table does not exist, you can create a table from the field definitions and index definitions using the CreateTable method. This property is read-only.

## TIBTable.Filter

[TIBTable](#)      [See also](#)

Specifies rows in a dataset that meet the filter conditions.

**property** Filter;

### Description

Use the Filter property to restrict the rows in the dataset to those that meet the filter conditions. The syntax for Filter is the same as the search condition that appears after a WHERE clause in a Select statement. For example, to view only those records where the value in the Country field contains 'France' or 'Fiji':

```
Country = 'France' or Country = 'Fiji'
```

You can use standard SQL wildcards such as percent (%) and underscore (\_) in the condition when you use the LIKE operator. The following filter condition retrieves all Countries beginning with 'F':

```
Country LIKE 'F%'
```

To view rows that have a NULL value in the Country column and Contact\_Name is not NULL, use the IS operator:

```
Country is NULL and Contact_Name is not NULL
```

You can also use complex expression in filter clauses this one that retrieves rows with Country values that use Francs as currency. This statement gets countries with currencies as 'BFranc', 'SFranc', and 'FFranc'

```
Country IN (SELECT Country from Country where Currency = '_Franc')
```

## TIBTable.Filtered

[TIBTable](#)      [See also](#)

Specifies whether or not filtering is active for a table.

**property** Filtered: Boolean;

### Description

Check Filtered to determine whether or not dataset filtering is in effect. If Filtered is True, then filtering is active. To apply filter conditions specified in the Filter property or the OnFilterRecord event handler, set Filtered to True.

## TIBTable.IndexDefs

[TIBTable](#)

[See also](#)

[Example](#)

Contains information about the indexes for a table.

**property** IndexDefs: TIndexDefs;

### Description

IndexDefs is an array of index items, each of which describes an available index for the table. Ordinarily an application accesses or specifies indexes through the IndexFieldNames or IndexFields properties. If IndexDefs is updated or manually edited, the StoreDefs property becomes True.

**Note:** The index items in IndexDefs may not always reflect the current indexes available for a table. Before examining IndexDefs, call its Update method to refresh the item list.

## TIBTable.IndexFieldCount

[TIBTable](#)

[See also](#)

[Example](#)

Indicates the number of fields that comprise the current key.

**property** IndexFieldCount: Integer;

### Description

Examine IndexFieldCount to determine the number of fields that comprise the current key. For indexes based on a single column, IndexFieldCount returns 1. For multi-column indexes, IndexFieldCount indicates the number of fields upon which the index is based.

## TIBTable.IndexFieldNames

[TIBTable](#)

[See also](#)

Lists the columns to use as an index for a table.

**property** IndexFieldNames: **String**;

### Description

Use IndexFieldNames as an alternative method of specifying the index to use for a table. In IndexFieldNames specify the name of each column to use as an index for a table. Ordering of column names is significant. Separate names with semicolon.

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

## TIBTable.IndexFields

[TIBTable](#)   [See also](#)   [Example](#)

Retrieves or sets a field for an index.

**property** IndexFields [Index: Integer]: TField;

### Description

IndexFields provides a zero-based array of field objects, each of which corresponds to a field in the current index. Index is an ordinal value indicating the position of a field in the index. The first field in the index is IndexFields[0], the second is IndexFields[1], and so on.

**Note:** Do not set IndexField directly. Instead use the IndexFieldNames property to order datasets on the fly at runtime.



## TIBTable.IndexName

[TIBTable](#)   [See also](#)   [Example](#)

Identifies an index for the table.

**property** IndexName: **String**;

### Description

Use IndexName to specify an index for a table. If IndexName is empty, a table's sort order is based on its primary index.

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

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

## TIBTable.MasterFields

[TIBTable](#)

[See also](#)

Specifies one or more fields in a master table to link with corresponding fields in this table in order to establish a master-detail relationship between the tables.

**property** MasterFields: String;

### Description

Use MasterFields after setting the MasterSource property to specify the names of one or more fields to use in establishing a detail-master relationship between this table and the one specified in MasterSource.

MasterFields is a string containing one or more field names in the master table. Separate field names with semicolons.

Each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

## TIBTable.MasterSource

[TIBTable](#)

[See also](#)

Specifies the name of the data source for a dataset to use as a master table in establishing a detail-master relationship between this table and another one.

**property** MasterSource: TDataSource;

### Description

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

After setting the MasterSource property, specify which fields to use in the master table by setting the MasterFields property. At runtime each time the current record in the master table changes, the new values in those fields are used to select corresponding records in this table for display.

## TIBTable.ReadOnly

[TIBTable](#)      [See also](#)

Specifies whether a table is read-only for this application.

**property** ReadOnly: Boolean;

### Description

Use the ReadOnly property to prevent users from updating, inserting, or deleting data in the table. By default, ReadOnly is False, meaning users can potentially alter a table's data.

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

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

- 1 Set the Active property to False.
- 2 Set ReadOnly to True.

## TIBTable.StoreDefs

[TIBTable](#)

[See also](#)

Indicates whether the table's field and index definitions persist with the data module or form.

**property** StoreDefs: Boolean;

### Description

If StoreDefs is True, the table's index and field definitions are stored with the data module or form.

Setting StoreDefs to True makes the CreateTable method into a one-step procedure that creates fields, indexes, and validity checks at runtime.

StoreDefs is False by default. It becomes True whenever FieldDefs or IndexDefs is updated or edited manually; to prevent edited (or imported) definitions from being stored, reset StoreDefs to False.

## TIBTable.TableTypes

[TIBTable](#)      [See also](#)

Sets the types of relations displayed in the TableName drop-down list.

```
type TIBTableType = (ttSystem, ttView);
```

```
    TIBTableTypes = set of TIBTableType
```

```
property TableTypes: TIBTableTypes;
```

### Description

Use TableTypes to change which types of relations are displayed in the TableName drop-down list, in addition to user tables. TableTypes are:

ttSystem          System tables and views

ttView            User views

## TIBTable.TableName

[TIBTable](#)      [See also](#)

Indicates the name of the database table or view that this component encapsulates.

```
property TableName: String;
```

### Description

Use TableName to specify the name of the database relation this component encapsulates. To set TableName to a meaningful value, the Database property should already be set. If Database is set at design time, then select a valid table name from the TableName drop-down list in the Object Inspector.

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

## TIBTable.TableNames

[TIBTable](#)      [See also](#)

Returns a list of table names.

**property** TableNames: **TStrings**;

### Description

The TableNames property is an internal property used to display a list of the table and view names in the database.

## TIBTable.UniDirectional

[TIBTable](#)      [See also](#)

Determines whether or not bidirectional cursors are enabled for a table.

**property** UniDirectional: Boolean;

### Description

Use UniDirectional to determine whether or not bidirectional cursors are enabled for a table.



## TIBTable.UpdateObject

[TIBTable](#)

[See also](#)

Specifies the update object component used to update a read-only result set when cached updates are enabled.

**property** UpdateObject;

### Description

Use UpdateObject to specify the TUpdateObject component to use in an application that must be able to update a read-only result set.

In a query made against multiple tables, a live result set cannot be returned. In these cases, UpdateObject can be used to specify a TIBUpdateSQL component that performs updates as a separate transaction that is transparent to the application.

## TIBTable events

[TIBTable](#)

[Alphabetically Legend](#)

### Derived from TIBCustomDataSet

- ▶ [OnUpdateError](#)
- ▶ [OnUpdateRecord](#)

### Derived from TDataSet

- ▶ [AfterCancel](#)
- ▶ [AfterClose](#)
- ▶ [AfterDelete](#)
- ▶ [AfterEdit](#)
- ▶ [AfterInsert](#)
- ▶ [AfterOpen](#)
- ▶ [AfterPost](#)
- ▶ [AfterRefresh](#)
- ▶ [AfterScroll](#)
- ▶ [BeforeCancel](#)
- ▶ [BeforeClose](#)
- ▶ [BeforeDelete](#)
- ▶ [BeforeEdit](#)
- ▶ [BeforeInsert](#)
- ▶ [BeforeOpen](#)
- ▶ [BeforePost](#)
- ▶ [BeforeRefresh](#)
- ▶ [BeforeScroll](#)
- ▶ [OnCalcFields](#)
- ▶ [OnDeleteError](#)
- ▶ [OnEditError](#)
- ▶ [OnNewRecord](#)
- ▶ [OnPostError](#)

## TIBTable events

TIBTable

By object

Legend

▶	<u>AfterCancel</u>
▶	<u>AfterClose</u>
▶	<u>AfterDelete</u>
▶	<u>AfterEdit</u>
▶	<u>AfterInsert</u>
▶	<u>AfterOpen</u>
▶	<u>AfterPost</u>
▶	<u>AfterRefresh</u>
▶	<u>AfterScroll</u>
▶	<u>BeforeCancel</u>
▶	<u>BeforeClose</u>
▶	<u>BeforeDelete</u>
▶	<u>BeforeEdit</u>
▶	<u>BeforeInsert</u>
▶	<u>BeforeOpen</u>
▶	<u>BeforePost</u>
▶	<u>BeforeRefresh</u>
▶	<u>BeforeScroll</u>
▶	<u>OnCalcFields</u>
▶	<u>OnDeleteError</u>
▶	<u>OnEditError</u>
▶	<u>OnNewRecord</u>
▶	<u>OnPostError</u>
▶	<u>OnUpdateError</u>
▶	<u>OnUpdateRecord</u>

## **TIBTable methods**

[TIBTable](#)

[Alphabetically](#)

### **In TIBTable**

[AddIndex](#)

[Create](#)

[CreateTable](#)

[DeleteIndex](#)

[DeleteTable](#)

[Destroy](#)

[EmptyTable](#)

[GetDetailLinkFields](#)

[GetIndexNames](#)

[GotoCurrent](#)

### **Derived from TIBCustomDataSet**

[ApplyUpdates](#)

[CancelUpdates](#)

[CreateBlobStream](#)

[FetchAll](#)

[GetCurrentRecord](#)

[GetFieldData](#)

[Locate](#)

[Lookup](#)

[RevertRecord](#)

[Translate](#)

[UpdateStatus](#)

### **Derived from TDataSet**

[ActiveBuffer](#)

[Append](#)

[AppendRecord](#)

[BookmarkValid](#)

[Cancel](#)

[CheckBrowseMode](#)

[ClearFields](#)

[Close](#)

[CompareBookmarks](#)

[ControlsDisabled](#)

[CursorPosChanged](#)

[Delete](#)

[DisableControls](#)

[Edit](#)

[EnableControls](#)

[FieldByName](#)

[FindField](#)

[FindFirst](#)

[FindLast](#)  
[FindNext](#)  
[FindPrior](#)  
[First](#)  
[GetBlobFieldData](#)  
[FreeBookmark](#)  
[GetBookmark](#)  
[GetDetailDataSets](#)  
[GetFieldList](#)  
[GetFieldNames](#)  
[GetProviderAttributes](#)  
[GotoBookmark](#)  
[Insert](#)  
[InsertRecord](#)  
[IsEmpty](#)  
[IsLinkedTo](#)  
[IsSequenced](#)  
[Last](#)  
[MoveBy](#)  
[Next](#)  
[Open](#)  
[Post](#)  
[Prior](#)  
[Refresh](#)  
[Resync](#)  
[SetFields](#)  
[UpdateCursorPos](#)  
[UpdateRecord](#)

**Derived from TComponent**

[DestroyComponents](#)  
[Destroying](#)  
[ExecuteAction](#)  
[FindComponent](#)  
[FreeNotification](#)  
[FreeOnRelease](#)  
[GetNamePath](#)  
[GetParentComponent](#)  
[HasParent](#)  
[InsertComponent](#)  
[RemoveComponent](#)  
[SafeCallException](#)  
[UpdateAction](#)

**Derived from TPersistent**

[Assign](#)

**Derived from TObject**AfterConstructionBeforeDestructionClassInfoClassNameClassNamelsClassParentClassTypeCleanupInstanceDefaultHandlerDispatchFieldAddressFreeFreeInstanceGetInterfaceGetInterfaceEntryGetInterfaceTableInheritsFromInitInstanceInstanceSizeMethodAddressMethodNameNewInstance

## TIBTable methods

[TIBTable](#)

[By object](#)

[ActiveBuffer](#)  
[AddIndex](#)  
[AfterConstruction](#)  
[Append](#)  
[AppendRecord](#)  
[ApplyUpdates](#)  
[Assign](#)  
[BeforeDestruction](#)  
[BookmarkValid](#)  
[Cancel](#)  
[CancelUpdates](#)  
[CheckBrowseMode](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[ClearFields](#)  
[Close](#)  
[CompareBookmarks](#)  
[ControlsDisabled](#)  
[Create](#)  
[CreateBlobStream](#)  
[CreateTable](#)  
[CursorPosChanged](#)  
[DefaultHandler](#)  
[Delete](#)  
[DeleteIndex](#)  
[DeleteTable](#)  
[Destroy](#)  
[DestroyComponents](#)  
[Destroying](#)  
[DisableControls](#)  
[Dispatch](#)  
[Edit](#)  
[EmptyTable](#)  
[EnableControls](#)  
[ExecuteAction](#)  
[FreeNotification](#)  
[FreeOnRelease](#)  
[FetchAll](#)

FieldByName  
FieldAddress  
FindComponent  
FindField  
FindFirst  
FindLast  
FindNext  
FindPrior  
First  
Free  
FreeBookmark  
FreeInstance  
GetBlobFieldData  
GetBookmark  
GetCurrentRecord  
GetDetailDataSets  
GetDetailLinkFields  
GetFieldData  
GetFieldList  
GetFieldNames  
GetIndexNames  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
GetNamePath  
GetParentComponent  
GetProviderAttributes  
GotoBookmark  
GotoCurrent  
HasParent  
InheritsFrom  
InitInstance  
Insert  
InsertComponent  
InsertRecord  
InstanceSize  
IsEmpty  
IsLinkedTo  
IsSequenced  
Last  
Locate  
Lookup  
MethodAddress  
MethodName  
MoveBy



Next

NewInstance

Open

Post

Prior

Refresh

RemoveComponent

Resync

RevertRecord

SafeCallException

SetFields

Translate

UpdateAction

UpdateCursorPos

UpdateRecord

UpdateStatus

## TIBTable.AddIndex

[TIBTable](#)

[See also](#)

[Example](#)

Creates a new index for the table.

```
procedure AddIndex(const Name, Fields: string; Options: TIndexOptions const  
    DescFields: string = '');
```

### Description

Call AddIndex to create a new index for the table associated with a dataset. The index created with this procedure is added to the database table underlying the dataset. Name is the name of the new index. Fields is a semicolon-delimited list of the fields to include in the index. Options is a potentially restricted set attributes for the index. It can specify that an index

- Represents the primary index for a dataset. (ixPrimary)
- Contains no duplicate values (ixUnique).
- Sorts records in descending order (isDescending).

**Warning:** Attempting to create an index using options that are not applicable to the table raises an exception.

## TIBTable.Create

[TIBTable](#)

[See also](#)

Creates an instance of a table component.

**constructor** Create (AOwner: TComponent) ;

### Description

Call Create to instantiate a table declared in an application if it was not placed on a form at design time. Create calls its inherited Create constructor, creates an empty index definitions list, creates an empty data link, and creates an empty list of index files.

## TIBTable.CreateTable

[TIBTable](#)

[See also](#)

[Example](#)

Builds a new table using new structure information.

**procedure** CreateTable;

### Description

Call CreateTable at runtime to create a table using this dataset's current definition. CreateTable overwrites an existing table's structure and data; to avoid overwriting an existing table, check Exists before calling CreateTable.

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

If the IndexDefs property contain values, these values are used to create index definitions for the dataset.

## TIBTable.DeleteIndex

[TIBTable](#)

[See also](#)

Deletes an index for the table.

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

### Description

Call DeleteIndex to remove an index for a table. Name is the name of the index to delete. DeleteIndex cannot remove an index used by a constraint.

## TIBTable.DeleteTable

[TIBTable](#)

[See also](#)

Deletes an existing database table.

**procedure** DeleteTable;

### Description

Call DeleteTable to delete an existing database table associated with the table component through its Database and TableName properties. A table must be closed before it can be deleted.

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

## TIBTable.Destroy

[TIBTable](#)

[See also](#)

Destroys the instance of a table.

**destructor** Destroy;

### Description

Do not call Destroy directly. Instead call Free to verify that the table is not already freed before calling Destroy. Destroy frees the index files list for the table, frees its data link, frees its index definitions, and then calls its inherited Destroy destructor.

## TIBTable.EmptyTable

[TIBTable](#)

[See also](#)

Deletes all records from the table.

**procedure** EmptyTable;

### Description

The EmptyTable method deletes all records from the database table specified by the Database and TableName properties.

**Note:** Deletion of records can fail if the user lacks sufficient privileges to perform the delete operation.



## TIBTable.GetDetailLinkFields

[TIBTable](#)

[See also](#)

Lists the field components that link this dataset as a detail of a master dataset.

**procedure** GetDetailLinkFields(MasterFields, DetailFields: TList);

### Description

GetDetailLinkFields fills two lists of TFields that define a master-detail relationship between this table and another (master) dataset. The MasterFields list is filled with fields from the master table whose values must equal the values of the fields in the DetailFields list. The DetailFields list is filled with fields from the calling dataset.

## **TIBTable.GetIndexNames**

[TIBTable](#)

[See also](#)

Retrieves a list of available indexes for a table.

**procedure** GetIndexNames (List: TStrings);

### **Description**

Call GetIndexNames to retrieve a list of all available indexes for a table. List is a string list object, created and maintained by the application, into which to retrieve the index names.

## TIBTable.GotoCurrent

[TIBTable](#)

[See also](#)

Synchronizes the current record for this table with the current record of a specified table component.

**procedure** GotoCurrent (Table: TIBTable) ;

### Description

Call GotoCurrent to synchronize the cursor position for this table based on the cursor position in another dataset that uses a different data source component, but which is connected to the same underlying database table. Table is the name of the table component whose cursor position to use for synchronizing.

**Note:** This procedure works only for table components that have the same Database and TableName properties. Otherwise an exception is raised.

GotoCurrent is mainly for use in applications that have two table components that are linked to the same underlying database table through different data source components. It enables an application to ensure that separate views of the data appear to be linked.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



TDataSet



TIBCustomDataSet



## TIBTransaction

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBTransaction provides discrete transaction control over a one or more database connections in a database application.

### Unit

IBDatabase

### Description

All TIBCustomDataSet descendants and TIBSQL need to use a transaction along with a database component to gain access to data in a database.

**Note:** In Midas applications, every query must be in its own transaction. You must use one transaction component for each query component.

## TIBTransaction properties

[TIBTransaction Alphabetically](#) [Legend](#)

### In TIBTransaction

- ▶ [Active](#)
- ▶▶ [DatabaseCount](#)
- ▶▶ [Databases](#)
- ▶▶ [DefaultAction](#)
- ▶▶ [DefaultDatabase](#)
- ▶▶ [Handle](#)
- ▶▶ [HandleIsShared](#)
- ▶▶ [IdleTimer](#)
- ▶▶ [InTransaction](#)
- ▶▶ [Params](#)
- ▶▶ [SQLObjectCount](#)
- ▶▶ [SQLObjects](#)
- ▶▶ [TPB](#)
- ▶▶ [TPBLength](#)

### Derived from TComponent

- ▶▶ [ComObject](#)
  - ▶▶ [ComponentCount](#)
  - ▶▶ [ComponentIndex](#)
  - ▶▶ [Components](#)
  - ▶▶ [ComponentState](#)
  - ▶▶ [ComponentStyle](#)
  - ▶▶ [DesignInfo](#)
- ▶▶ [Name](#)
- ▶▶ [Owner](#)
- ▶▶ [Tag](#)
- ▶▶ [VCLComObject](#)



## TIBTransaction properties

[TIBTransaction By object](#)

[Legend](#)

- ▶ [Active](#)
- ▶ [ComObject](#)
- ▶ [ComponentCount](#)
- ▶ [ComponentIndex](#)
- ▶ [Components](#)
- ▶ [ComponentState](#)
- ▶ [ComponentStyle](#)
- ▶ [DatabaseCount](#)
- ▶ [Databases](#)
- ▶ [DefaultAction](#)
- ▶ [DefaultDatabase](#)
- ▶ [DesignInfo](#)
- ▶ [Handle](#)
- ▶ [HandleIsShared](#)
- ▶ [IdleTimer](#)
- ▶ [InTransaction](#)
- ▶ [Name](#)
- ▶ [Owner](#)
- ▶ [Params](#)
- ▶ [SQLObjectCount](#)
- ▶ [SQLObjects](#)
- ▶ [Tag](#)
- ▶ [DefaultAction](#)
- ▶ [TPB](#)
- ▶ [TPBLength](#)
- ▶ [VCLComObject](#)

## **TIBTransaction.Active**

[TIBTransaction](#) [See also](#)

Specifies whether or not a transaction is active.

**property** Active: Boolean;

### **Description**

Use Active to determine or set a transaction's active state.

## **TIBTransaction.DatabaseCount**

[TIBTransaction](#) [See also](#)

Indicates the number of databases that are part of the transaction.

**property** DatabaseCount: Integer;

### **Description**

Use DatabaseCount to determine the number of databases involved in a transaction.

## TIBTransaction.Databases

[TIBTransaction See also](#)

Returns the database at the given integer index.

**property** Databases[**Index**: Integer]: TIBDatabase;

### Description

Use Databases to return the database at the given integer index.

## TIBTransaction.DefaultAction

[TIBTransaction See also](#)

Specifies what action a transaction should take upon timing out.

**type** TTransactionAction = (taRollback, taCommit, taRollbackRetaining, taCommitRetaining);

**property** DefaultAction: TTransactionAction;

### Description

Use DefaultAction to what action the transaction should take when the IdleTimer limit is met. The transaction action can be one of the following:

taRollback	Rolls back the transaction
taCommit	Commits the transaction
taRollbackRetaining	Rolls back the transaction, but retains the current transaction context
	<b>Note:</b> You must install InterBase 6 to use this feature.
taCommitRetaining	Commits the transaction, but retains the current transaction context

## TIBTransaction.DefaultDatabase

[TIBTransaction See also](#)

Sets or returns the default database for the transaction.

**property** DefaultDatabase: TIBDatabase;

### Description

Use DefaultDatabase to set or return the default database for the transaction.

## TIBTransaction.Handle

[TIBTransaction](#) [See also](#)

Returns the transaction handle.

**property** Handle: TISC\_TR\_HANDLE;

### Description

Use Handle to retrieve a handle to the transaction. Handle is assigned an initial value when a transaction is started.

## TIBTransaction.HandleIsShared

[TIBTransaction](#) See also

Indicates whether or not a the handle is shared.

**property** HandleIsShared: Boolean;

### Description

HandleIsShared returns True when the transaction handle is shared by more than one transaction component.

## TIBTransaction.IdleTimer

[TIBTransaction](#) [See also](#)

Specifies how long the transaction should wait before automatically committing or rolling back.

**property** IdleTimer: Integer;

### Description

Use IdleTimer to indicate how long a transaction should be allowed to remain idle before automatically committing or rolling back the data. Use DefaultAction to determine which action the transaction should take.



## TIBTransaction.InTransaction

[TIBTransaction](#) See also

Indicates whether a database transaction is in progress or not.

**property** InTransaction: Boolean;

### Description

Examine InTransaction at run-time to determine if a database transaction is currently in progress.

InTransaction is True if a transaction is in progress, False otherwise.

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

## TIBTransaction.Params

[TIBTransaction](#) [See also](#)

Returns the transaction parameter buffer associated with the transaction component.

**property** Params: TStrings;

### Description

Use Params to examine and set parameters in the transaction parameter buffer. Refer to the Interbase API Guide for the names of the parameters to provide.

## TIBTransaction.SQLObjectCount

[TIBTransaction](#) [See also](#)

Returns the number of active datasets associated with the database component.

**property** `SQLObjectCount: Integer;`

### Description

Use the `SQLObjectCount` property to return the number currently active InterBase datasets, TIBSQL objects, and Blobs associated with the database component. As SQL objects are opened and closed, this value changes appropriately.

## TIBTransaction.SQLObjects

[TIBTransaction](#) [See also](#)

Provides an indexed array of all active datasets for a database component.

**property** SQLObjects[Index: Integer]: TIBBase;

### Description

Use the SQLObjects to access active InterBase datasets, TIBSQL objects, and Blobs associated with the database component.

## **TIBTransaction.TPB**

[TIBTransaction](#) [See also](#)

Provides a read-only view of the transaction parameter buffer.

**property** TPB: **PChar**;

### **Description**

Use TPB view the transaction parameter buffer. To write to the transaction parameter buffer, use the Params property.

## TIBTransaction.TPBLength

[TIBTransaction](#) [See also](#)

Returns the length of the transaction parameter buffer.

**property** TPBLength: Short;

### Description

Use TPBLength to retrieve the length of the transaction parameter buffer

## TIBTransaction events

[TIBTransaction Alphabetically](#) [Legend](#)

In TIBTransaction

▶ [OnIdleTimer](#)

TIBTransaction events

[TIBTransactionBy object](#)   [Legend](#)



OnIdleTimer



## TIBTransaction.OnIdleTimer

[TIBTransaction](#) See also

Occurs after a transaction has timed out.

**property** OnIdleTimer: TNotifyEvent;

### Description

Write an OnIdleTimer event handler to take specific actions after a transaction is allowed to remain idle for the number of seconds specified by IdleTimer.

## **TIBTransaction methods**

### [TIBTransaction Alphabetically](#)

#### **In TIBTransaction**

[AddDatabase](#)

[Call](#)

[CheckDatabasesInList](#)

[CheckInTransaction](#)

[CheckNotInTransaction](#)

[Commit](#)

[CommitRetaining](#)

[Create](#)

[Destroy](#)

[FindDatabase](#)

[RemoveDatabase](#)

[RemoveDatabases](#)

[Rollback](#)

[RollbackRetaining](#)

[StartTransaction](#)

#### **Derived from TComponent**

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

#### **Derived from TPersistent**

[Assign](#)

#### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

Dispatch  
FieldAddress  
Free  
FreeInstance  
GetInterface  
GetInterfaceEntry  
GetInterfaceTable  
InheritsFrom  
InitInstance  
InstanceSize  
MethodAddress  
MethodName  
NewInstance

## **TIBTransaction methods**

### [TIBTransaction By object](#)

[AddDatabase](#)

[AfterConstruction](#)

[Assign](#)

[BeforeDestruction](#)

[Call](#)

[CheckDatabasesInList](#)

[CheckInTransaction](#)

[CheckNotInTransaction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Commit](#)

[CommitRetaining](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecuteAction](#)

[FieldAddress](#)

[FindComponent](#)

[FindDatabase](#)

[Free](#)

[FreeInstance](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

NewInstance

RemoveComponent

RemoveDatabase

RemoveDatabases

Rollback

RollbackRetaining

SafeCallException

UpdateAction

StartTransaction

## TIBTransaction.AddDatabase

[TIBTransaction See also](#)

Associates a database to the transaction.

**function** AddDatabase (db: TIBDatabase) : Integer;

### Description

Call AddDatabase to associate a database to the transaction.

## TIBTransaction.Call

[TIBTransaction](#) [See also](#)

Returns an error message based on the error code.

**procedure** Call(ErrCode: ISC\_STATUS; RaiseError: Boolean): ISC\_STATUS;

### Description

Call is an internal method used to make calls to the InterBase API, and gives you the option of raising an exception or returning an error based on the value of RaiseError.

## **TIBTransaction.CheckDatabasesInList**

[TIBTransaction See also](#)

Checks for databases in the list.

**procedure** CheckDatabasesInList;

### **Description**

Call CheckDatabasesInList to check if there are any databases in the list. If there are no databases in the list, an exception is raised.



## **TIBTransaction.CheckInTransaction**

[TIBTransaction](#) [See also](#)

Checks whether the transaction is active and whether there are any databases in the transaction's database list.

**procedure** CheckInTransaction;

### **Description**

Call CheckInTransaction to determine whether the transaction is active and whether there are any databases in the transaction's database list. If either condition is False, an exception is raised.

## **TIBTransaction.CheckNotInTransaction**

[TIBTransaction](#) [See also](#)

Checks that the transaction is not active and that there are no databases in the transaction's database list.

**procedure** `CheckNotInTransaction;`

### **Description**

Call `CheckInTransaction` to determine that the transaction is not active and that there are no databases in the transaction's database list. If either condition is False, an exception is raised.

## TIBTransaction.Commit

[TIBTransaction](#) [See also](#) [Example](#)

Permanently stores updates, insertions, and deletions of data associated with the current transaction, and ends the current transactions.

**procedure** Commit;

### Description

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

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

## TIBTransaction.CommitRetaining

[TIBTransaction](#) See also

Commits the active transaction and retains the transaction context after a commit.

**procedure** CommitRetaining;

### Description

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

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

## TIBTransaction.Create

[TIBTransaction](#) [See also](#)

Creates an instance of a transaction component.

**constructor** `Create (AOwner: TComponent);`

### Description

Call Create to instantiate a transaction component at runtime. An application creates a transaction component in order to control the component's existence and set its properties and events.

Create instantiates a transaction component and creates an empty string list for the Params property.

## TIBTransaction.Destroy

[TIBTransaction](#) [See also](#)

Destroys the instance of the transaction component.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead, call Free, which verifies that the transaction component is not already freed before calling Destroy.

Destroy disconnects from the database server, if necessary. It then frees the string resources allocated for the Params and SQLObjects properties before calling its inherited destructor.

## TIBTransaction.FindDatabase

[TIBTransaction See also](#)

Finds the index of the associated database.

**function** FindDatabase (db: TIBDatabase): Integer;

### Description

Call FindDatabase to find the index of the associated database.

## **TIBTransaction.RemoveDatabase**

[TIBTransaction See also](#)

Disassociates a database from the transaction.

**procedure** RemoveDatabase (Idx: Integer);

### **Description**

Call RemoveDatabase to disassociate a specified database from the transaction.



## **TIBTransaction.RemoveDatabases**

[TIBTransaction](#) [See also](#)

Disassociates all databases from the transaction.

**procedure** RemoveDatabases;

### **Description**

Call RemoveDatabases to disassociate all databases from the transaction.

## TIBTransaction.Rollback

[TIBTransaction](#) [See also](#) [Example](#)

Cancels all updates, insertions, and deletions for the current transaction and ends the transaction.

**procedure** Rollback;

### Description

Call Rollback to cancel all updates, insertions, and deletions for the current transaction and to end the transaction. The current transaction is the last transaction started by calling StartTransaction.

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

## TIBTransaction.RollbackRetaining

[TIBTransaction See also](#)

Cancels all updates, insertions, and deletions for the current transaction and retains the transaction context.

**procedure** RollbackRetaining;

### Description

Call RollbackRetaining to roll back to the database server all updates, insertions, and deletions of data associated with the current transaction and then retain the transaction context. The current transaction is the last transaction started by calling StartTransaction.

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

**Note:** You must install InterBase 6 to use this feature.

## **TIBTransaction.StartTransaction**

[TIBTransaction](#) [See also](#) [Example](#)

Begins a new transaction against the database server.

**procedure** StartTransaction;

### **Description**

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

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

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent

## IBAlloc procedure

[See also](#)

Allocates or de-allocates memory for a given size, and initializes the new memory to zero.

### Unit

IB

```
procedure IBAlloc(var P; OldSize, NewSize: Integer);
```

Use IBAlloc to allocate or de-allocate memory for a given size, and initialize the new memory to zero.



## CheckStatusVector function

[See also](#)

Sets or unsets status vector checking.

### Unit

IB

**function** CheckStatusVector(ErrorCodes: array of ISC\_STATUS): Boolean;

Set CheckStatusVector to True to enable status vector checking.

## FreeIBTLGlobals procedure

[See also](#)

Frees the allocated thread-local storage.

### Unit

IB

**procedure** FreeIBTLGlobals;

### Description

Use the FreeIBTLGlobals procedure to free the allocated thread-local storage.

## InitializeIBTLGlobals procedure

[See also](#)

Initializes the thread-local storage.

### Unit

IB

**procedure** InitializeIBTLGlobals;

### Description

Use the InitializeIBTLGlobals procedure to initialize thread-local storage.

Thread-local storage is used to store global variables pertaining to a thread. The global variables are defined in TIBTLGlobals. Currently, it contains the status vector.

## IBError procedure

[See also](#)

Returns the error message for the specified error code.

### Unit

IB

```
procedure IBError(ErrMess: TIBClientError; const Args: array of const);
```

### Description

Use the IBError procedure to look up the appropriate error message for a specified error code and raise an EIBClientError exception.

## **IBDataBaseError procedure**

[See also](#)

Interprets the SQLCode and IBCode status and passes it to the EIBInterBaseError exception.

### **Unit**

IB

**procedure** IBDataBaseError;

### **Description**

Use the IBDatabaseError procedure to look at the status vector, interpret the SQLCode and IBCode status, construct the status string, and pass it to the EIBInterBaseError exception.

## SetIBDataBaseErrorMessages procedure

[See also](#)

Enables you to choose the error messages you want to see.

### Unit

IB

**procedure** SetIBDataBaseErrorMessages (Value: TIBDataBaseErrorMessages);

### Description

Use the SetIBDataBaseErrorMessages procedure to choose the error messages that you want to use. The available options are defined in TIBDataBaseErrorMessages.

## GetIBDataBaseErrorMessages function

[See also](#)

Returns the current error message setting.

### Unit

IB

```
function GetIBDataBaseErrorMessages: TIBDataBaseErrorMessages;
```

### Description

Use the GetIBDataBaseErrorMessages function to return the current error message setting.

## StatusVector function

[See also](#)

Retrieves the thread-specific status vector from thread-local storage.

### Unit

IB

```
function StatusVector: PISC_STATUS;
```

### Description

Use the StatusVector function to retrieve the thread-specific status from the thread-local storage.



## StatusVectorArray function

[See also](#)

Returns a pointer to the status vector array.

### Unit

IB

```
function StatusVectorArray: PStatusVector;
```

### Description

Use the StatusVectorArray function to return a pointer to the status vector array from the thread-local storage.

## StatusVectorAsText function

[See also](#)

Gets the status vector and returns it as a formatted string.

### Unit

IB

```
function StatusVectorAsText: string;
```

### Description

Use the StatusVector function to retrieve the status vector and return it as a formatted string.

## LoadIBInstallLibrary procedure

[See also](#)

Loads IBInstall.DLL into memory.

### Unit

IBIntf

**procedure** LoadIBInstallLibrary;

### Description

Use the LoadIBInstallLibrary procedure to load the IBInstall.DLL into memory and resolve the respective entry points into to the respective procedure pointer in the unit.

## LoadIBLibrary procedure

[See also](#)

Loads GDS32.DLL into memory.

### Unit

IBIntf

**procedure** LoadIBLibrary;

### Description

Use the LoadIBLibrary procedure to load the GDS32.DLL into memory and resolve the respective entry points into the respective procedure pointer in the unit. In addition, it loads the InterBase 6.0 entry points, if they are available.

## FreeIBInstallLibrary procedure

[See also](#)

Frees the IBInstall.DLL from memory.

### Unit

IBIntf

**procedure** FreeIBInstallLibrary;

### Description

Use the FreeIBInstallLibrary procedure to free the previously loaded the IBInstall.DLL from memory.

## FreeIBLibrary procedure

[See also](#)

Frees the GDS32.DLL from memory.

### Unit

IBIntf

**procedure** FreeIBLibrary;

### Description

Use the FreeIBInstallLibrary procedure to free the previously loaded the GDS32.DLL from memory.

## CheckIBInstallLoaded procedure

[See also](#)

Attempts to load IBInstall.DLL into memory if it is not already loaded.

### Unit

IBIntf

**procedure** CheckIBInstallLoaded;

### Description

Use the CheckIBInstallLibrary procedure to load IBInstall.DLL into memory if it is not already loaded. If unsuccessful, this procedure raises an exception.

## CheckIBLoaded procedure

[See also](#)

Attempts to load GDS32.DLL into memory if it is not already loaded.

### Unit

IBIntf

**procedure** CheckIBLoaded;

### Description

Use the CheckIBInstallLibrary procedure to load GDS32.DLL into memory if it is not already loaded. If unsuccessful, this procedure raises an exception.



## GetIBClientVersion function

[See also](#)

Returns the major version number of the InterBase client.

### Unit

IBIntf

```
function GetIBClientVersion: Integer;
```

### Description

Use the GetIBClientVersion function to retrieve the major version number of the InterBase client.

## DisableMonitoring procedure

[See also](#)

Disables SQL monitoring.

### Unit

IBSQLMonitor

**procedure** DisableMonitoring;

### Description

Use the DisableMonitoring procedure to disable SQL monitoring.

## EnableMonitoring procedure

[See also](#)

Enables SQL monitoring.

### Unit

IBSQLMonitor

**procedure** EnableMonitoring;

### Description

Use the EnableMonitoring procedure to enable SQL monitoring.

## MonitoringEnabled function

[See also](#)

Indicates whether or not monitoring is enabled.

### Unit

IBSQLMonitor

**function** MonitoringEnabled: Boolean;

### Description

Use the MonitoringEnabled function to indicate whether or not monitoring is enabled.

## MonitorHook function

[See also](#)

Returns the reference to the global monitor hook.

### Unit

IBSQLMonitor

**function** MonitorHook: TIBSQLMonitorHook;

### Description

Use the MonitorHook function to return the reference to the global monitor hook. If the monitor hook does not exist, it is created.

## GenerateDPB procedure

[See also](#)

Populates a database parameter block with the values supplied by a TStrings object.

### Unit

IBDatabase

```
procedure GenerateDPB(sl: TStrings; var DPB: string; var DPBLength: Short);
```

### Description

Use the GenerateDPB procedure to populate a database parameter block (DPB) with the values supplied by a TStrings object. For more information on the DPB format, refer to the InterBase API Guide.

## GenerateTPB procedure

[See also](#)

Populates a transaction parameter block with the values supplied by a TStrings object.

### Unit

IBDatabase

```
procedure p GenerateTPB(sl: TStrings; var TPB: string; var TPBLength: Short);
```

### Description

Use the GenerateTPB procedure to populate a transaction parameter block (TPB) with the values supplied by a TStrings object. For more information on the TPB format, refer to the InterBase API Guide.



## TIBUpdateSQL

[Hierarchy](#)

[Properties](#)

[Methods](#)

[See also](#)

TIBUpdateSQL provides an object for updating read-only datasets when cached updates are enabled.

### Unit

IBUpdateSQL

### Description

Use a TIBUpdateSQL object to provide SQL statements used to update read-only datasets represented by TIBQuery components when cached updates are enabled. A dataset is read-only either by design or circumstance. If a dataset is read-only by design, the application itself does not provide a user interface for updating data, but may institute a programmatic scheme behind the scenes.

TIBUpdateSQL provides a mechanism for circumventing what some developers consider an SQL-92 limitation. It enables a developer to provide INSERT, UPDATE, DELETE, and REFRESH statements for performing separate update queries on otherwise read-only result sets in such a manner that the separate update queries are transparent to the end user.

In practical application, a TIBUpdateSQL object is placed on a data module or form, and linked to a TIBQuery component through that component's UpdateObject property. If the UpdateObject property points to a valid TIBUpdateSQL object, the SQL statements belonging to the update object are automatically applied when cached updates are applied.



## TIBUpdateSQL properties

[TIBUpdateSQL Alphabetically](#) [Legend](#)

### In TIBUpdateSQL

- ▶ [DataSet](#)
- ▶ [DeleteSQL](#)
- ▶ [InsertSQL](#)
- ▶ [ModifySQL](#)
- ▶ [Query](#)
- ▶ [RefreshSQL](#)
- ▶ [SQL](#)

### Derived from TComponent

- ▶ [ComObject](#)
- ▶ [ComponentCount](#)
- ▶ [ComponentIndex](#)
- ▶ [Components](#)
- ▶ [ComponentState](#)
- ▶ [ComponentStyle](#)
- ▶ [DesignInfo](#)
- ▶ [Name](#)
- ▶ [Owner](#)
- ▶ [Tag](#)
- ▶ [VCLComObject](#)

## TIBUpdateSQL properties

[TIBUpdateSQL By object](#)

[Legend](#)

▶	<u>ComObject</u>
▶	<u>ComponentCount</u>
	<u>ComponentIndex</u>
▶	<u>Components</u>
▶	<u>ComponentState</u>
▶	<u>ComponentStyle</u>
▶	<u>DataSet</u>
▶	<u>DeleteSQL</u>
	<u>DesignInfo</u>
▶	<u>InsertSQL</u>
▶	<u>ModifySQL</u>
▶	<u>Name</u>
	▶ <u>Owner</u>
	▶ <u>Query</u>
▶	<u>RefreshSQL</u>
	<u>SQL</u>
▶	<u>Tag</u>
	<u>VCLComObject</u>

## **TIBUpdateSQL.DataSet**

[TIBUpdateSQL See also](#)

Identifies the dataset to which a TIBUpdateSQL component belongs.

**property** DataSet;

### **Description**

At design time, setting the dataset object's UpdateObject property automatically sets the DataSet property of the specified TIBUpdateSQL object. An application should only need to set this property if it creates a new update component at run time.

## TIBUpdateSQL.DeleteSQL

[TIBUpdateSQL See also](#)

Specifies the SQL DELETE statement to use when applying a cached deletion of a record.

**property** DeleteSQL: TStrings;

### Description

Set DeleteSQL to the SQL DELETE statement to use when applying a deletion to a record. Statements can be parameterized queries. To create a DELETE statement at design time, use the UpdateSQL editor to create statements, such as"

```
delete from Employee
where
    Emp_No = :OLD_Emp_No
```

At run time, an application can write a statement directly to this property to set or change the DELETE statement.

**Note:** As the example illustrates, DeleteSQL supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, use the field name with the prefix 'OLD\_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

## TIBUpdateSQL.InsertSQL

[TIBUpdateSQL See also](#)

Specifies the SQL INSERT statement to use when applying a cached insertion of a record.

**property** InsertSQL: TStrings;

### Description

Set InsertSQL to the SQL INSERT statement to use when applying an insertion to a dataset. Statements can be parameterized queries. To create a INSERT statement at design time, use the UpdateSQL editor to create statements, such as"

```
insert into Country  
(Country, Currency)  
values (:Country, :Currency)
```

At run time, an application can write a statement directly to this property to set or change the INSERT statement.

## TIBUpdateSQL.ModifySQL

[TIBUpdateSQL See also](#)

Specifies the SQL UPDATE statement to use when applying an update to a record and cached updates is enabled.

**property** ModifySQL: TStrings;

### Description

Set ModifySQL to the SQL UPDATE statement to use when applying an updated record to a dataset. Statements can be parameterized queries. To create a UPDATE statement at design time, use the UpdateSQL editor to create statements, such as:

```
update Employee  
set Last_Name = :Last_Name  
where Emp_No = :OLD_Emp_No
```

At run time, an application can write a statement directly to this property to set or change the UPDATE statement.

**Note:** As the example illustrates, ModifySQL supports an extension to normal parameter binding. To retrieve the value of a field as it exists prior to application of cached updates, the field name with 'OLD\_'. This is especially useful when doing field comparisons in the WHERE clause of the statement.

## TIBUpdateSQL.Query

[TIBUpdateSQL](#) See also

Returns the query object used to perform a specified kind of update.

**type** TUpdateKind = (ukModify, ukInsert, ukDelete)

**property** Query[UpdateKind: TUpdateKind!ALink(TUpdateKind\_Type,1)]: TIBQuery!  
ALink(TIBQuery\_Object,1);

### Description

Query returns the TIBQuery object used to perform a particular form of SQL update. UpdateKind specifies which query object to retrieve. UpdateKind can be one of the following:

Value	Meaning
ukModify	Return the query object used to execute UPDATE statements
ukInsert	Return the query object used to execute INSERT statements
ukDelete	Return the query object used to execute DELETE statements

Each query object executes a particular kind of SQL statement. The contents of the SQL statements executed by these objects can be accessed directly using the ModifySQL, InsertSQL, and DeleteSQL properties.

The main purpose of Query is to provide a way for an application to set the properties for an update query object or to call the query object's methods.

**Note:** If a particular kind of update statement is not provided, then its corresponding query object is nil. For example, if an application does not provide an SQL statement for the DeleteSQL property, then Query[ukDelete] returns nil.

## TIBUpdateSQL.RefreshSQL

[TIBUpdateSQL See also](#)

Specifies the SQL SELECT statement to use when refreshing a dataset.

**property** RefreshSQL: TStrings;

### Description

Set RefreshSQL to the SQL SELECT statement to use when refreshing a dataset. Statements can be parameterized queries. To create a SELECT statement at design time, use the UpdateSQL editor to create statements, such as"

```
SELECT Country, Currency FROM Country WHERE  
Country = :Country
```

At run time, an application can write a statement directly to this property to set or change the SELECT statement.



## TIBUpdateSQL.SQL

[TIBUpdateSQL See also](#)

Returns a specified SQL statement used when applying cached updates.

**type** TUpdateKind = (ukModify, ukInsert, ukDelete)

**property** SQL[UpdateKind: TUpdateKind!ALink(TUpdateKind\_Type,1)]: TStrings!  
ALink(TStrings\_Object,1);

### Description

Returns the SQL statement in the ModifySQL, InsertSQL, or DeleteSQL property, depending on the setting of UpdateKind. UpdateKind can be any of the following:

Value	Meaning
ukModify	Return the SQL statement used to update records in the dataset
ukInsert	Return the SQL statement used to insert new records into the dataset
ukDelete	Return the SQL statement used to delete records in the dataset

## **TIBUpdateSQL methods**

[TIBUpdateSQL Alphabetically](#)

### **In TIBUpdateSQL**

[Apply](#)

[Create](#)

[Destroy](#)

[ExecSQL](#)

[SetParams](#)

### **Derived from TComponent**

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### **Derived from TPersistent**

[Assign](#)

[GetNamePath](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNames](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

MethodAddress

MethodName

NewInstance

## TIBUpdateSQL methods

### [TIBUpdateSQL By object](#)

[AfterConstruction](#)

[Apply](#)

[Assign](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[DestroyComponents](#)

[Destroying](#)

[Dispatch](#)

[ExecSQL](#)

[ExecuteAction](#)

[FieldAddress](#)

[FindComponent](#)

[Free](#)

[FreeInstance](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InheritsFrom](#)

[InitInstance](#)

[InsertComponent](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[RemoveComponent](#)

[SafeCallException](#)

[SetParams](#)

[UpdateAction](#)

## TIBUpdateSQL.Apply

[TIBUpdateSQL](#) [See also](#)

Sets the parameters for a specified SQL statement type, and executes the resulting statement.

**type** TUpdateKind = (ukModify, ukInsert, ukDelete)

**procedure** Apply(UpdateKind: TUpdateKind);

### Description

Call Apply to set parameters for an SQL statement and execute it to update a record. UpdateKind indicates which SQL statement to bind and execute, and can be one of the following values:

Value	Meaning
ukModify	Bind and execute the SQL statement in the ModifySQL property
ukInsert	Bind and execute the SQL statement in the InsertSQL property
ukDelete	Bind and execute the SQL statement in the DeleteSQL property

Apply is primarily intended for manually executing update statements from an OnUpdateRecord event handler.

**Note:** If an SQL statement does not contain parameters, it is more efficient to call ExecSQL instead of Apply.

## TIBUpdateSQL.Create

[TIBUpdateSQL See also](#)

Creates an instance of an update object.

**constructor** Create (AOwner: TComponent) ;

### Description

Call Create to instantiate an update object at run time. You do not need to call Create for update objects placed in a data module or form at design time. Delphi automatically handles these objects.

## TIBUpdateSQL.Destroy

[TIBUpdateSQL](#) [See also](#)

Frees an instance of an update object.

**constructor** Destroy;

### Description

Do not call Destroy directly in an application. Usually destruction of update objects is handled automatically by Delphi. If an application creates its own instance of an update object, however, the application should call Free, which verifies that the update object is not already freed before calling Destroy.

## TIBUpdateSQL.ExecSQL

[TIBUpdateSQL](#) See also

Executes a specified type of SQL statement to perform an update for an otherwise read-only results set when cached updates is enabled.

**type** TUpdateKind = (ukModify, ukInsert, ukDelete)

**procedure** ExecSQL(UpdateKind: TUpdateKind);

### Description

Call ExecSQL to execute the SQL statement necessary for updating the records belonging to a read-only result set when cached updates is enabled. UpdateKind specifies the statement to execute, and can be one of the following values:

Value	Meaning
ukModify	Execute the SQL statement used to update records in the dataset
ukInsert	Execute the SQL statement used to insert new records into the dataset
ukDelete	Execute the SQL statement used to delete records in the dataset.

If the statement to execute contains any parameters, an application must call SetParams to bind the parameters before calling ExecSQL. To determine if a statement contains parameters, examine the appropriate ModifySQL, InsertSQL, or DeleteSQL property, depending on the statement type intended for execution.

**Note:** To both bind parameters and execute a statement, call Apply.



## TIBUpdateSQL.SetParams

[TIBUpdateSQL See also](#)

Binds parameters in an SQL statement prior to statement execution.

**type** TUpdateKind = (ukModify, ukInsert, ukDelete)

**procedure** SetParams (UpdateKind: TUpdateKind);

### Description

Call SetParams to bind parameters in an SQL statement associated with the update object prior to executing the statement. UpdateKind indicates the type of statement for which to bind parameters, and can be one of the following values:

Value	Meaning
ukModify	Bind parameters for the SQL statement used to update records
ukInsert	Bind parameters for the SQL statement used to insert new records
ukDelete	Bind parameters for the SQL statement used to delete records

Parameters are indicated in an SQL statement by a colon. Except for the leading colon in the parameter name, the parameter name must exactly match the name of an existing field name for the dataset.

**Note:** Parameter names can be prefaced by the 'OLD\_' indicator. If so, the old value of the field is used to perform the update instead of any updates in the cache.

## Scope



Published

## Accessibility



Read-only

## Hierarchy

TObject



TPersistent



TComponent



TDataSetUpdateObject

## TIBXSQLDA

[Hierarchy](#)   [Properties](#)   [Methods](#)[See also](#)

TIBXSQLDA provides properties and methods for use with the IBSQL component.

### Unit

IBSQL

### Description

Use a TIBXSQLDA object to provide properties and methods for use with the IBSQL component.

All Dynamic SQL (DSQL) applications must declare one or more extended descriptor areas (XSQLDAs). The XSQLDA is a host language data structure that DSQL uses to transport data to or from a database when processing an SQL statement string. There are two types of XSQLDAs: input descriptors and output descriptors. Both input and output descriptors are implemented using the XSQLDA structure.

## TIBXSQLDA properties

[TIBXSQLDA](#) [Alphabetically](#) [Legend](#)

### In TIBXSQLDA

#### ▸ AsXSQLDA

	<u>Count</u>
▸	<u>Modified</u>
▸	<u>Names</u>
▸	<u>RecordSize</u>
▸	<u>UniqueRelationName</u>
▸	<u>Vars</u>

## TIBXSQLDA properties

[TIBXSQLDA](#)

[By object](#)

[Legend](#)

### ▸ AsXSQLDA

- Count
- Modified
- Names
- RecordSize
- UniqueRelationName
- Vars

## TIBXSQLDA.AsXSQLDA

[TIBXSQLDA](#) [See also](#)

Represents the XSQLDA field's value as an XSQLDA value.

**property** AsXSQLDA: PXSQLDA;

### Description

Use the AsXSQLDA property to read the value of the field's data into an object or variable of type XSQLDA, or to assign an XSQLDA value to the contents of the field.

## **TIBXSQLDA.Count**

[TIBXSQLDA](#) [See also](#)

Returns the number of XSQLDA fields.

**property** Count: Integer;

### **Description**

Use the Count property to return the number of XSQLDA fields.



## **TIBXSQLDA.Modified**

[TIBXSQLDA](#) [See also](#)

Indicates whether a field has been modified.

**property** Modified: Boolean;

### **Description**

Use the Modified property to determine whether a field has been modified.

## **TIBXSQLDA.Names**

[TIBXSQLDA](#) [See also](#)

Returns the XSQLDA field names.

**property** Names: String;

### **Description**

Use the Names property to return the XSQLDA field names.

## **TIBXSQLDA.RecordSize**

[TIBXSQLDA](#) [See also](#)

Returns the size of the XSQLDA record.

**property** RecordSize: Integer;

### **Description**

Use the RecordSize property to return the XSQLDA record size.

## **TIBXSQLDA.UniqueRelationName**

[TIBXSQLDA](#) [See also](#)

Returns the name of the unique relation.

**property** UniqueRelationName: String;

### **Description**

Use the UniqueRelationName property to return the name of the relation if only one relation is involved in the query. Otherwise, it returns nil. This property is primarily used for internal purposes.

## **TIBXSQLDA.Vars**

[TIBXSQLDA](#) [See also](#)

Returns the XSQLVAR defined for the XSQLDA parameter.

**property** Vars: [Idx: Integer]: TIBXSQLVAR;

### **Description**

Use XSQLVAR to return the XSQLVAR defined for the XSQLDA parameter.

## **TIBXSQLDA methods**

[TIBXSQLDA](#) [Alphabetically](#)

### **In TIBXSQLDA**

[AddName](#)

[ByName](#)

[Create](#)

[Destroy](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBXSQLDA methods

[TIBXSQLDA](#) [By object](#)

[AddName](#)

[AfterConstruction](#)

[BeforeDestruction](#)

[ByName](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## **TIBXSQLDA.AddName**

[TIBXSQLDA](#) [See also](#)

Adds a name to the XSQLDA structure.

**procedure** AddName (FieldName: String; Idx: Integer);

### **Description**

Call AddName to add a name to the XSQLDA structure.



## TIBXSQLDA.ByName

[TIBXSQLDA](#) [See also](#)

Returns the XSQLVAR fields by name.

**function** ByName: [Idx: String]: TIBXSQLVAR;

### Description

Use the ByName function to return the XSQLVAR fields by name.

## **TIBXSQLDA.Create**

[TIBXSQLDA](#) [See also](#)

Creates an instance of an XSQLDA structure.

**constructor** Create (Query: TIBSQL) ;

### **Description**

Call Create to create an instance of an XSQLDA structure. Create queries the DynamicSQL component for the structure members.

## **TIBXSQLDA.Destroy**

[TIBXSQLDA](#) [See also](#)

Destroys the XSQLDA structure.

**destructor** Destroy;

### **Description**

Do not call Destroy directly. Call Free instead. Free checks to ensure that the object instance is not nil before calling Destroy.

## Scope



Published

## Accessibility



Read-only

**Hierarchy**  
TObject

## TIBXSQLVAR

[Hierarchy](#)   [Properties](#)   [Methods](#)[See also](#)

TIBXSQLVAR provides properties and methods for use with the IBSQL component.

### Unit

IBSQL

### Description

Use a TIBXSQLVAR object to provide properties and methods for use with the IBSQL component.

The XSQLVAR structure is a field, sqlvar, in the XSQLDA. The sqlvar is especially important, because one XSQLVAR must be defined for each input parameter or column returned.

Applications do not declare instances of the XSQLVAR ahead of time, but must, instead, dynamically allocate storage for the proper number of XSQLVAR structures required for each DSQL statement before it is executed, then deallocate it, as appropriate, after statement execution.

**TIBXSQLVAR properties**

[TIBXSQLVAR](#) [Alphabetically](#) [Legend](#)

**In TIBXSQLVAR**

<u>AsCurrency</u>	
<u>AsDate</u>	
<u>AsDateTime</u>	
<u>AsDouble</u>	
<u>AsFloat</u>	
<u>AsInt64</u>	
<u>AsInteger</u>	
<u>AsLong</u>	
<u>AsPointer</u>	
<u>AsQuad</u>	
<u>AsShort</u>	
<u>AsString</u>	
<u>AsVariant</u>	
<u>AsXSQLVAR</u>	
<u>Data</u>	
▀ <u>Index</u>	
	<u>IsNull</u>
	<u>IsNullable</u>
	<u>Modified</u>
<u>Name</u>	
▀ <u>Size</u>	
▀	<u>SQLType</u>
	<u>AsTime</u>
	<u>Value</u>

TIBXSQLVAR properties

[TIBXSQLVAR](#) [By object](#) [Legend](#)

<u>AsCurrency</u>	
<u>AsDate</u>	
<u>AsDateTime</u>	
<u>AsDouble</u>	
<u>AsFloat</u>	
<u>AsInt64</u>	
<u>AsInteger</u>	
<u>AsLong</u>	
<u>AsPointer</u>	
<u>AsQuad</u>	
<u>AsShort</u>	
<u>AsString</u>	
<u>AsVariant</u>	
<u>AsXSQLVAR</u>	
<u>Data</u>	
▀ <u>Index</u>	
	<u>IsNull</u>
	<u>IsNullable</u>
	<u>Modified</u>
<u>Name</u>	
▀ <u>Size</u>	
▀	<u>SQLType</u>
	<u>AsTime</u>
	<u>Value</u>



## TIBXSQLVAR.AsCurrency

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a Currency value.

**property** AsCurrency: Currency;

Use the AsCurrency property to read the value of the XSQLVAR field's data into an object or variable of type Currency, or to assign a Currency value to the contents of the field.

## **TIBXSQLVAR.AsDate**

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a TDateTime value.

**property** AsDate: TDateTime;

### **Description**

Use the AsDate property to read the value of the field's data into an object or variable of type TDateTime, or to assign a TDateTime value to the contents of the field. The Time portion of the TDateTime value is set to zero.

## **TIBXSQLVAR.AsDateTime**

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a TDateTime value.

**property** AsDateTime: TDateTime;

### **Description**

Use the AsDateTime property to read the value of the field's data into an object or variable of type TDateTime, or to assign a TDateTime value to the contents of the field.

## TIBXSQLVAR.AsDouble

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a Double value.

**property** AsDouble: Double;

### Description

Use the AsDouble property to read the value of the field's data into an object or variable of type Double, or to assign a Double value to the contents of the field.

## **TIBXSQLVAR.AsFloat**

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a Float value.

**property** AsFloat: Float;

### **Description**

Use the AsDouble property to read the value of the field's data into an object or variable of type Float, or to assign a Float value to the contents of the field.

## TIBXSQLVAR.AsInt64

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a 64-bit integer.

**property** AsInt64: Int64;

### Description

Use the AsInt64 property to read the value of the field's data into a 64-bit integer, or to assign an 64-bit integer value to the contents of the field.

## TIBXSQLVAR.AsInteger

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a 32-bit integer value.

**property** AsInteger: AsInteger;

### Description

Use the AsInteger property to read the value of the field's data into an integer, or to assign an integer value to the contents of the field.

## **TIBXSQLVAR.AsLong**

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a Long integer value.

**property** AsLong: Long;

### **Description**

Use the AsLong property to read the value of the field's data into a Long integer, or to assign a Long integer value to the contents of the field.



## **TIBXSQLVAR.AsPointer**

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a Pointer value.

**property** AsPointer: Pointer;

### **Description**

Use the AsPointer property to read the value of the field's data into an object or variable of type Pointer, or to assign a Pointer value to the contents of the field.

## TIBXSQLVAR.AsQuad

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a Quad value.

**property** AsQuad: TISC\_QUAD;

### Description

Use the AsQuad property to read the value of the field's data into an object or variable of type Quad, or to assign a Quad value to the contents of the field.

## TIBXSQLVAR.AsShort

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a Short integer value.

**property** AsShort: Short;

### Description

Use the AsShort property to read the value of the field's data into a Short integer, or to assign a Short integer value to the contents of the field.

## TIBXSQLVAR.AsString

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a string.

**property** AsString: String;

### Description

Use the AsString property to read the value of the fields data into a String, or to assign a String value to the contents of the field.

## TIBXSQLVAR.AsTime

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as a TDateTime value.

**property** AsTime: TDateTime;

### Description

Use the AsTime property to read the value of the field's data into an object or variable of type TDateTime, or to assign a TDateTime value to the contents of the field. The Date portion of the field is set to zero.

## **TIBXSQLVAR.AsVariant**

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as type Variant.

**property** AsVariant: Variant;

### **Description**

Use the AsVariant property to read the value of the field's data into a Variant, or to assign a Variant value to the contents of the field.

## TIBXSQLVAR.AsXSQLVAR

[TIBXSQLVAR](#) [See also](#)

Represents the XSQLVAR field's value as an XSQLVAR value.

**property** AsXSQLVAR: PXSQLVAR;

### Description

Use the AsXSQLVAR property to read the value of the field's data into an object or variable of type XSQLVAR, or to assign an XSQLVAR value to the contents of the field.

## **TIBXSQLVAR.Data**

[TIBXSQLVAR](#) [See also](#)

Accesses the underlying InterBase XSQLVAR structure.

**property** Data: PSQLVAR;

### **Description**

Use the Data property to access the underlying InterBase XSQLVAR structure.



## **TIBXSQLVAR.Index**

[TIBXSQLVAR](#) [See also](#)

Indicates the position of the XSQLVAR in the XSQLDA.

**property** Index: Integer;

### **Description**

Use the Index property to obtain the position of the XSQLVAR in the XSQLDA.

## TIBXSQLVAR.IsNull

[TIBXSQLVAR](#) [See also](#)

Indicates whether the field has a value assigned to it.

**property** IsNull: Boolean;

### Description

Use the IsNull property to determine if the field contains a value. If IsNull is True, the field is blank. If IsNull is False, the field has a value.

## TIBXSQLVAR.IsNullable

[TIBXSQLVAR](#) [See also](#)

Indicates whether the field can have a value assigned to it.

**property** IsNullable: Boolean;

### Description

Use the IsNullable property to determine if the field can contain a value. If IsNullable is True, the field can contain a value. If IsNullable is False, the field cannot contain a value.

## **TIBXSQLVAR.Modified**

[TIBXSQLVAR](#) [See also](#)

Indicates whether a field has been modified.

**property** Modified: Boolean;

### **Description**

Use the Modified property to determine whether a field has been modified.

## **TIBXSQLVAR.Name**

[TIBXSQLVAR](#) [See also](#)

Returns the name of the XSQLVAR.

**property** Name: String;

### **Description**

Use the Name property to return the name of the XSQLVAR.

## **TIBXSQLVAR.Size**

[TIBXSQLVAR](#) [See also](#)

Indicates the maximum size, in bytes, of data in the sqldata field of the XSQLVAR.

**property** Size: Integer;

### **Description**

Use the Size property to return the maximum size, in bytes, of data in the sqldata field of the XSQLVAR.

## **TIBXSQLVAR.SQLType**

[TIBXSQLVAR](#) [See also](#)

Indicates the SQL datatype of parameters or select-list items.

**property** SQLType: Integer;

### **Description**

Read the SQLType property to indicate the SQL datatype of parameters or select-list items.

## **TIBXSQLVAR.Value**

[TIBXSQLVAR](#) [See also](#)

Returns the value of the XSQLVAR field component.

**property** Value: Variant;

### **Description**

Use Value to return the value of the XSQLVAR field component as a Variant.



## **TIBXSQLVAR methods**

[TIBXSQLVAR](#) [Alphabetically](#)

### **In TIBXSQLVAR**

[Assign](#)

[Create](#)

[LoadFromFile](#)

[LoadFromStream](#)

[SaveToFile](#)

[SaveToStream](#)

### **Derived from TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Destroy](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

## TIBXSQLVAR methods

[TIBXSQLVAR](#) [By object](#)

[AfterConstruction](#)

[Assign](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[Create](#)

[DefaultHandler](#)

[Destroy](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

[InstanceSize](#)

[LoadFromFile](#)

[LoadFromStream](#)

[MethodAddress](#)

[MethodName](#)

[NewInstance](#)

[SafeCallException](#)

[SaveToFile](#)

[SaveToStream](#)

## **TIBXSQLVAR.Assign**

[TIBXSQLVAR](#) [See also](#)

Assigns another XSQLVAR to the XSQLVAR component.

**procedure** `Assign(Source: TIBXSQLVAR);`

### **Description**

Use Assign to assign another XSQLVAR to the XSQLVAR component. The fields must have compatible data types.

## **TIBXSQLVAR.Create**

[TIBXSQLVAR](#) [See also](#)

Creates an instance of an XSQLVAR structure.

**constructor** Create (Query: TIBSQL) ;

### **Description**

Call Create to create an instance of an XSQLVAR structure based on a query to the TIBSQL component.

## **TIBXSQLVAR.LoadFromFile**

[TIBXSQLVAR](#) [See also](#)

Loads the contents of a file to a Blob field.

**procedure** LoadFromFile(const FileName: String);

### **Description**

Call LoadFromFile to load the contents of a file to a Blob field.

## **TIBXSQLVAR.LoadFromStream**

[TIBXSQLVAR](#) [See also](#)

Loads a stream into a Blob field.

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

### **Description**

Call LoadFromStream to load a stream into a Blob field.

## **TIBXSQLVAR.SaveToFile**

[TIBXSQLVAR](#) [See also](#)

Saves the contents of a Blob field to a file.

**procedure** SaveToFile(const FileName: String);

### **Description**

Call SaveToFile to save the contents of a Blob field to a file.

## **TIBXSQLVAR.SaveToStream**

[TIBXSQLVAR](#) [See also](#)

Saves the contents of a Blob field to a stream.

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

### **Description**

Call SaveToStream to save the contents of a Blob field to a stream.



## Scope



Published

## Accessibility



Read-only

**Hierarchy**  
TObject



## TIBEvents

[Hierarchy](#)

[Properties](#)

[Methods](#)

[Events](#)

[See also](#)

TIBEvents provides a method for applications to respond to posted events.

### Unit

IBEvents

### Description

Use a TIBEvents component to allow your application to register interest in, and asynchronously handle, events posted by an InterBase server. The InterBase event allows applications to respond to actions and database changes made by other, concurrently running applications, without having to resort to polling the database on a regular basis, or communicating directly with the other applications.

In essence, the TIBEvents component allows an application to say 'I want to be informed when events X, Y and Z occur.' When any of the requested events does occur, the InterBase server notifies the application and OnEventAlert is called.

## TIBEvents properties

[TIBEvents](#)   [Alphabetically Legend](#)

### In TIBEvents

- ▶ [Database](#)
- ▶ [Events](#)
- ▶ [Queued](#)
- ▶ [Registered](#)

### Derived from TComponent

- ▶ [ComObject](#)
  - ▶ [ComponentCount](#)
  - ▶ [ComponentIndex](#)
  - ▶ [Components](#)
  - ▶ [ComponentState](#)
  - ▶ [ComponentStyle](#)
  - ▶ [DesignInfo](#)
- ▶ [Name](#)
- ▶ [Owner](#)
- ▶ [Tag](#)
- ▶ [VCLComObject](#)

TIBEvents properties

[TIBEvents](#)   [By object](#)   [Legend](#)

▶ <u>ComObject</u>	
▶	<u>ComponentCount</u>
	<u>ComponentIndex</u>
▶	<u>Components</u>
▶	<u>ComponentState</u>
▶	<u>ComponentStyle</u>
▶	<u>Database</u>
<u>DesignInfo</u>	
▶	<u>Events</u>
▶	<u>Name</u>
▶	<u>Owner</u>
▶	<u>Queued</u>
▶	<u>Registered</u>
▶	<u>Tag</u>
	<u>VCLComObject</u>

## TIBEvents.Database

[TIBEvents](#)   [See also](#)

Specifies the database on which to perform event alerter tasks.

**property** Database: TIBDatabase;

### Description

Use Database to specify the database on which to perform event alerter tasks.

## TIBEvents.Events

[TIBEvents](#)   [See also](#)

Specifies the events to which TIBEvents responds.

**property** Events: TStrings;

### Description

Use the Events property to list events for which the IBEvents component will respond. A single IBEvents component can handle up to 15 events.

To add an event use the following code:

```
IBEvents.Events.Add('EVENT_NAME')
```

**Note:** Event names are case-sensitive.

## TIBEvents.Queued

[TIBEvents](#)   [See also](#)

Indicates that events are queued.

**property** Queued: Boolean;

### Description

Use Queued to determine if events are queued.



## TIBEvents.Registered

[TIBEvents](#)    [See also](#)

Indicates whether or not the event is registered.

**property** Registered: Boolean;

### Description

Use Registered to indicate whether events are registered. Set Registered to True to call RegisterEvents, which registers the events listed by the Events property.

## TIBEvents events

[TIBEvents](#)

[Alphabetically Legend](#)

In TIBEvents



[OnEventAlert](#)

TIBEvents events

[TIBEvents](#)   [By object](#)   [Legend](#)

 OnEventAlert

## TIBEvents.OnEventAlert

[TIBEvents](#)   [See also](#)

Occurs when an InterBase event is received.

```
property OnEventAlert: TEventAlert;  
TEventAlert = procedure ( Sender: TObject; EventName: String; EventCount:  
    longint; var CancelAlerts: Boolean)
```

### Description

Write an OnEventAlert event handler to take specific actions when an InterBase event is received. EventName contains the name of the most recently received event. EventCount contains the number of EventName events received since OnEventAlert was last called.

Set CancelAlerts to True to cancel interest in any further events. To start receiving events again, call the QueueEvents method. You cannot call RegisterEvents, UnRegisterEvents, QueueEvents or CancelEvents from within an OnEventAlert event handler.

OnEventAlert runs as a separate thread to allow for true asynchronous event processing, however, the IBEvents component provides synchronization code to ensure that only one OnEventAlert event handler executes at any one time.

## **TIBEvents methods**

[TIBEvents](#)    [Alphabetically](#)

### **In TIBEvents**

[CancelEvents](#)

[Create](#)

[Destroy](#)

[QueueEvents](#)

[RegisterEvents](#)

[UnRegisterEvents](#)

### **Derived from TComponent**

[DestroyComponents](#)

[Destroying](#)

[ExecuteAction](#)

[FindComponent](#)

[FreeNotification](#)

[FreeOnRelease](#)

[GetNamePath](#)

[GetParentComponent](#)

[HasParent](#)

[InsertComponent](#)

[RemoveComponent](#)

[SafeCallException](#)

[UpdateAction](#)

### **Derived from (TPersistent**

[Assign](#)

### **Derived from (TObject**

[AfterConstruction](#)

[BeforeDestruction](#)

[ClassInfo](#)

[ClassName](#)

[ClassNamels](#)

[ClassParent](#)

[ClassType](#)

[CleanupInstance](#)

[DefaultHandler](#)

[Dispatch](#)

[FieldAddress](#)

[Free](#)

[FreeInstance](#)

[GetInterface](#)

[GetInterfaceEntry](#)

[GetInterfaceTable](#)

[InheritsFrom](#)

[InitInstance](#)

InstanceSize  
MethodAddress  
MethodName  
NewInstance

## TIBEvents methods

[TIBEvents](#)   [By object](#)

[AfterConstruction](#)  
[Assign](#)  
[BeforeDestruction](#)  
[CancelEvents](#)  
[ClassInfo](#)  
[ClassName](#)  
[ClassNamels](#)  
[ClassParent](#)  
[ClassType](#)  
[CleanupInstance](#)  
[Create](#)  
[DefaultHandler](#)  
[Destroy](#)  
[DestroyComponents](#)  
[Destroying](#)  
[Dispatch](#)  
[ExecuteAction](#)  
[FieldAddress](#)  
[FindComponent](#)  
[Free](#)  
[FreeInstance](#)  
[FreeNotification](#)  
[FreeOnRelease](#)  
[GetInterface](#)  
[GetInterfaceEntry](#)  
[GetInterfaceTable](#)  
[GetNamePath](#)  
[GetParentComponent](#)  
[HasParent](#)  
[InheritsFrom](#)  
[InitInstance](#)  
[InsertComponent](#)  
[InstanceSize](#)  
[MethodAddress](#)  
[MethodName](#)  
[NewInstance](#)  
[QueueEvents](#)  
[RegisterEvents](#)  
[RemoveComponent](#)  
[SafeCallException](#)  
[UnRegisterEvents](#)  
[UpdateAction](#)

## **TIBEvents.CancelEvents**

[TIBEvents](#)   [See also](#)

Cancels interest in pending events.

**procedure** CancelEvents;

### **Description**

Call CancelEvents to cancel interest in any pending InterBase events. CancelEvents does not unregister the events; call QueueEvents to restore interest in the events again.



## TIBEvents.Create

[TIBEvents](#)   [See also](#)

Creates an instance of TIBEvents.

**constructor** Create (AOwner: TComponent) ;

### Description

Call Create to create an instance of TIBEvents.

## TIBEvents.Destroy

[TIBEvents](#)   [See also](#)

Destroys an instance of TIBEvents.

**destructor** Destroy;

### Description

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the TIBEvents object is not already freed and only then calls Destroy. Destroy unregisters the events and then frees them.

## TIBEvents.QueueEvents

[TIBEvents](#)   [See also](#)

Starts event notification for the application.

**procedure** `QueueEvents;`

### Description

Call QueueEvents to allow your application to start receiving event notifications.

You must call RegisterEvents to specify which events you wish to receive before calling QueueEvents. If RegisterEvents has not been called an exception will be raised.

## **TIBEvents.RegisterEvents**

[TIBEvents](#)    [See also](#)

Registers interest in the events listed by the Events property.

**procedure** RegisterEvents;

### **Description**

Call RegisterEvents to register interest in the events listed in the Events property with the InterBase Event Manager. RegisterEvents calls the QueueEvents method to start receiving event notifications.

## **TIBEvents.UnRegisterEvents**

[TIBEvents](#)    [See also](#)

Unregisters interest in the events listed by the Events property.

**procedure** UnRegisterEvents;

### **Description**

Call UnRegisterEvents to unregister interest in the events in the Events list. UnregisterEvents calls CancelEvents to cancel any pending event notifications. When the IBEvents component is destroyed, UnRegisterEvents is called automatically.

## Scope



Published

## Accessibility



Read-only

**Scope**



Published

## Hierarchy

TObject



TPersistent



TComponent



## IBDatabase Editor dialog box

The Database Editor dialog box sets up the properties of a database that specify the connection that should be made to a database. This dialog box allows you to specify the type of database, the connection parameters, the user name, SQL role, and password, and whether or not a login prompt is required.

These properties of the database component, as well as others, can also be specified using the Object inspector.

To display the Database Editor dialog box, double click on an IBDatabase component.

### Dialog box options

#### Connection

Option	Meaning
Local	Indicates that the database is on the local server. Enables the Browse button, allowing you to search for the database with a Open File dialog.
Remote	Indicates that the database is on a remote server. Activates the Protocol and Server fields
Protocol	Sets the protocol for attaching to the remote server. The protocol can be TCP/IP, Named Pipe, or SPX.
Server	The name of the remote server.
Database	The name of the database.

#### Database Parameters

Option	Meaning
User Name	The name of the database user.
Password	The password for the database user.
SQLRole	The SQLRole name used to connect to the database.
Character Set	The character set used to connect to the database.
Login Prompt	Indicates whether a login prompt is required to access the database.
Settings	Displays the current parameters and allows you to add other parameters.

For example:

```
user_name=sysdba
password=masterkey
sql_role_name=finance
lc_ctype=WIN1252
```

For more information on database parameters, see the InterBase 6 API Guide.

## **IBTransaction Editor dialog box**

The Transaction Editor dialog box allows you to set up transaction parameters. This dialog box gives you four default transaction settings, which you can then customize if you wish. Once you modify the default transaction, the radio button is unset.

For a complete list of all the InterBase transaction parameters, refer to “Working with Transactions” in the InterBase 6 API Guide.

These properties of the transaction component, as well as others, can also be specified using the Object inspector.

To display the Transaction Editor dialog box, double click on an IBTransaction component. The following four choices are displayed:

### **Snapshot**

By default, Snapshot is set to concurrency and nowait, which means that the transaction is aware of other transactions, and does not wait for locks to be released, returning an error instead.

### **Read Committed**

By default, Read Committed is set to read\_committed, rec\_version, and nowait, which means that the transaction reads changes made by concurrent transactions, can read the most recently committed version of a transaction, and does not wait for locks to be released, returning an error instead.

### **Read-Only Table Stability**

By default, Read-Only Table Stability is set to read and consistency, which means that the transaction can read a specified table and locks out other transactions.

### **Read-Write Table Stability**

By default, Read-Write Table Stability is set to write and consistency, which means that the transaction can read and write to a specified table and locks out other transactions.

For a complete list of all the InterBase transaction parameters, refer to “Working with Transactions” in the InterBase 6 API Guide.

## IBUpdateSQL Editor dialog box

Use the Update SQL editor to create SQL statements for updating a dataset.

The TIBUpdateSQL object must be associated with a TIBQuery object by setting the TIBQuery property UpdateObject to the name of the TIBUpdateSQL object used to contain the SQL statements. A datasource, and database name must be selected for the TIBQuery object. In addition, the SQL property must include an SQL statement defining a table.

To open the SQL editor:

1. Select the TIBUpdateSQL object in the form.
2. Right-click and choose Update SQL editor.

The Update SQL editor has two pages, the Options page and the SQL page.

### The Options page

The Options page is visible when you first invoke the editor.

Table Name	Use the Table Name combo box to select the table to update. When you specify a table name, the Key Fields and Update Fields list boxes are populated with available columns.
Key Fields	The Key Fields list box is used to specify the columns to use as keys during the update. Generally the columns you specify here should correspond to an existing index.
Update Fields	The Update Fields list box indicates which columns should be updated. When you first specify a table, all columns in the Update Fields list box are selected for inclusion. You can multi-select fields as desired.
Get Table Fields	Read the table fields for the table name entered and list the fields.
Dataset Defaults	Use this button to restore the default values of the associated dataset. This will cause all fields in the Key Fields list and the Update Fields list to be selected and the table name to be restored.
Select Primary Keys	Click the Primary Key button to select key fields based on the primary index for a table.
Generate SQL	After you specify a table, select key columns, and select update columns, click the Generate SQL button to generate the preliminary SQL statements to associate with the update component's ModifySQL, InsertSQL, DeleteSQL, and RefreshSQL properties.
Quote Identifiers	Check the box labeled Quote Field Names to specify that all field names in generated SQL be enclosed by quotation marks. This option is disabled in pre-InterBase 6 databases.

### The SQL page

To view, modify, and refresh the generated SQL statements, select the SQL page. If you have generated SQL statements, then when you select this page, the statement for the ModifySQL property is already displayed in the SQL Text memo box. You can edit the statement in the box as desired.

**Note:** Keep in mind that generated SQL statements are intended to be starting points for creating update statements. You may need to modify these statements to make them execute correctly. Test each of the statements directly yourself before accepting them.

Use the Statement Type radio buttons (Modify, Insert, Delete, or Refresh) to switch among generated SQL statements and edit them as desired.

To accept the statements and associate them with the update component's SQL properties, click OK.

## AddIndex Example

In the example below, the AddIndex method is used to create an index named NewIndex. This index is based on two fields from the associated table, CustNo and CustName. The index NewIndex incorporates two index options through the TIndexOptions constants ixUnique and ixCaseInsensitive.

```
IBTable1.AddIndex('NewIndex', 'CustNo;CustName', [ixUnique,  
    ixCaseInsensitive]);
```

## BeforeInsert, Insert, AsInteger, FieldByName Example

This example uses the BeforeInsert event to do data validation; if the StrToInt function raises an exception, the edit control's contents are set to a valid value so the assignment to the INTEGER field in the table will succeed.

```
procedure TForm1.Table1BeforeInsert(DataSet: TDataSet);  
begin  
    try  
        {Make sure edit field can be converted to integer --  
        this will raise an exception if it can't }  
        StrToInt(Edit1.Text);  
    except  
        Edit1.Text := '0';  
    end;  
end;  
  
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    Table1.Insert;  
    Table1.FieldByName('QUANTITY').AsInteger := StrToInt(Edit1.Text);  
    Table1.Post;  
end;
```

## GetBookmark, GotoBookmark, FreeBookmark, FindPrior, Value, OnDataChange, BOF Example

This example uses a button to copy the value of a field in the previous record into the corresponding field in the current record.

```
procedure TForm1.CopyDataClick(Sender: TObject);  
var  
    SavePlace: TBookmark;  
    PrevValue: Variant;  
begin  
    with Table1 do  
        begin  
            { get a bookmark so that we can return to the same record }  
            SavePlace := GetBookmark;  
            { move to prior record }  
            FindPrior;  
            { get the value }  
            PrevValue := Fields[0].Value;  
            {Move back to the bookmark  
            this may not be the next record anymore  
            if something else is changing the dataset asynchronously }  
            GotoBookmark(SavePlace);  
            { Set the value }  
            Fields[0].Value := PrevValue;  
            { Free the bookmark }  
            FreeBookmark(SavePlace);  
        end;  
end;
```

To ensure that the button is disabled when there is no previous record, the OnDataChange event of the DataSource detects when the user moves to the beginning of file (BOF property becomes True), and disables the button.

```
procedure TForm1.Table1DataChange(Sender: TObject; Field: TField);  
begin  
    if Table1.BOF then  
        CopyData.Enabled := False  
    else  
        CopyData.Enabled := True;  
end;
```

## StartTransaction, Commit, Rollback, RollbackRetaining example

The following procedure illustrates how to apply a dataset's cached updates to a database in response to a button click:

```
procedure TForm1.ApplyButtonClick(Sender: TObject);  
begin  
    with CustomerQuery do  
        begin  
            IBDatabase1.Open;  
            IBTransaction1.StartTransaction;  
            Table1.Insert;  
            Table1.FieldName('QUANTITY').AsInteger := StrToInt(Edit1.Text);  
            Table1.Post;  
            IBTransaction1.Commit;  
        end;  
end;
```

In the above example, you could substitute Rollback or RollbackRetaining (an InterBase 6 feature) for Commit.

## Create,CreateBlobStream Example

The following example copies the data in the Notes field of IBTable1 to the Remarks field of IBTable2.

```
procedure TForm1.Button1Click(Sender: TObject);  
var  
    Stream1, Stream2 : TIBBlobStream;  
begin  
    Stream1 := TIBBlobStream.Create(IBTable1Notes, bmRead);  
    try  
        IBTable2.Edit;  
        { here's a different way to create a blob stream }  
        Stream2 := IBTable2.CreateBlobStream(IBTable2.FieldName('Remarks'),  
bmReadWrite);  
        try  
            Stream2.CopyFrom(Stream1, Stream1.Size);  
            IBTable2.Post;  
        finally  
            Stream2.Free;  
        end;  
    finally  
        Stream1.Free;  
    end;  
end;
```



## Table Creation Example

The following example shows how to create a table.

```
{ Don't overwrite an existing table }
if not IBTable1.Exists then begin
  with IBTable1 do begin
    { The Table component must not be active }
    Active := False;
    { First, describe the type of table and give }
    { it a name }
    DatabaseName := 'IBDataBase1';
    TableName := 'CustInfo';
    { Next, describe the fields in the table }
    with FieldDefs do begin
      Clear;
      with AddFieldDef do begin
        Name := 'Field1';
        DataType := ftInteger;
        Required := True;
      end;
      with AddFieldDef do begin
        Name := 'Field2';
        DataType := ftString;
        Size := 30;
      end;
    end;
    { Next, describe any indexes }
    with IndexDefs do begin
      Clear;
      with AddIndexDef do begin
        Name := '';
        Fields := 'Field1';
        Options := [ixPrimary];
      end;
      with AddIndexDef do begin
        Name := 'Fld2Indx';
      end;
    end;
    { Call the CreateTable method to create the table }
    CreateTable;
  end;
end;
```

## Database Example

```
{ Do a transaction }  
with Table1.Database do  
begin  
    StartTransAction;  
{ Post some records with Table1 }  
    Commit;  
end;
```

## **Prepare, ExecProc Example**

The following code executes the stored procedure:

```
IBStoredProc1.Params[0].AsString := Edit1.Text;  
IBStoredProc1.Prepare;  
IBStoredProc1.ExecProc;
```

## Prepare, Prepared Example

```
if not IBDataSet1.Prepared then
begin
    IBDataSet1.Close;
    IBDataSet1.Prepare;
    IBDataSet1.Open
end;
```

## **Prepare, Prepared Example**

```
if not IBSQL1.Prepared then  
begin  
    IBSQL1.Close;  
    IBSQL1.Prepare;  
    IBSQL1.Open  
end;
```

## SQL, ExecSQL Example

```
IBQuery1.Close;  
IBQuery1.SQL.Clear;  
IBQuery1.SQL.Add('Delete from Country where Name = ''Argentina'');  
IBQuery1.ExecSQL;
```

## ParamByName, GetData, GetDataSize Example

```
var Buffer: Pointer;  
begin  
    { Allocate enough space to hold the CustNo data }  
    GetMem(Buffer, IBQuery1.ParamByName('CustNo').GetDataSize);  
    try  
        { Retrieve the data }  
        IBQuery1.ParamByName('CustNo').GetData(Buffer);  
        { now do something with the data }  
    finally  
        FreeMem(MyBuffer);  
    end;  
end;
```

## ParamByName Example

```
StoredProc1.ParamByName('DNO').AsString := Edit1.Text;  
StoredProc1.Prepare;  
StoredProc1.ExecProc;  
Edit2.Text := FloatToStr(StoredProc1.ParamByName('TOT').AsFloat;
```



## GetStoredProcNames Example

```
MyStringList := TStringList.Create;  
try  
    Session.GetStoredProcNames('IB_EMPLOYEE', MyStringList);  
    { fill a list box with stored procedure names  
      for the user to select from }  
    ListBox1.Items := MyStringList;  
finally  
    MyStringList.Free;  
end;
```

## IndexDefs, IndexName

This example uses the IndexName property to sort the records in a table on the CustNo and OrderNo fields.

```
IBTable1.Active := False;
{ Get the current available indicies }
IBTable1.IndexDefs.Update;
{ Find one which combines Customer Number ('CustNo') and Order Number
  ('OrderNo') }
for I := 0 to IBTable1.IndexDefs.Count - 1 do
  if Table1.IndexDefs.Items[I].Fields = 'CustNo;OrderNo' then
    { set that index as the current index for the table }
    IBTable1.IndexName := IBTable1.IndexDefs.Items[I].Name;
IBTable1.Active := True;
```

## IndexFields, IndexFieldCount Example

The following code calculates the total length of the index and assigns it to the variable TotalLen.

```
TotalLen := 0;  
with IBTable1 do  
  for I := 0 to IndexFieldCount - 1 do  
    Inc(TotalLen, IndexFields[I].DataSize);
```

## Mode, AbortOnKeyViol, Execute, MovedCount, KeyViolCount Example

The following code uses the BatchMove component to add records to a table. After the records have been added, the number of new records is reported on the status line.

```
with BatchMove1 do  
begin  
    Mode := batAppend;  
    AbortOnKeyViol := False;  
    Execute;  
    StatusBar1.SimpleText := IntToStr(MovedCount - KeyViolCount) + ' records  
    added';  
end;
```

## ParamCount, DataType, StrToIntDef, AsXXX Example

This example fills in the parameters of a query from the entries of a list box.

```
var
  I: Integer;
  ListItem: string;
begin
  for I := 0 to IBQuery1.ParamCount - 1 do
    begin
      ListItem := ListBox1.Items[I];
      case IBQuery1.Params[I].DataType of
        ftString:
          IBQuery1.Params[I].AsString := ListItem;
        ftSmallInt:
          IBQuery1.Params[I].AsSmallInt := StrToIntDef(ListItem, 0);
        ftInteger:
          IBQuery1.Params[I].AsInteger := StrToIntDef(ListItem, 0);
        ftWord:
          IBQuery1.Params[I].AsWord := StrToIntDef(ListItem, 0);
        ftBoolean:
          begin
            if ListItem = 'True' then
              IBQuery1.Params[I].AsBoolean := True
            else
              IBQuery1.Params[I].AsBoolean := False;
            end;
        ftFloat:
          IBQuery1.Params[I].AsFloat := StrToFloat(ListItem);
        ftCurrency:
          IBQuery1.Params[I].AsCurrency := StrToFloat(ListItem);
        ftBCD:
          IBQuery1.Params[I].AsBCD := StrToCurr(ListItem);
        ftDate:
          IBQuery1.Params[I].AsDate := StrToDate(ListItem);
        ftTime:
          IBQuery1.Params[I].AsTime := StrToTime(ListItem);
        ftDateTime:
          IBQuery1.Params[I].AsDateTime := StrToDateTime(ListItem);
      end;
    end;
  end;
end;
```

## ParamCount, Params, ParamType Example

```
{ Set all input parameters to an empty string }  
with IBStoredProc1 do  
  for I := 0 to ParamCount - 1 do  
    if (Params[I].ParamType = ptInput) or  
      (Params[I].ParamType = ptInputOutput) then  
      Params[I].AsString := '';
```

## Params Example

The following code runs an insert query to add a record for Lichtenstein into the country table.

```
IBQuery2.SQL.Clear;
IBQuery2.SQL.Add('INSERT INTO COUNTRY (NAME, CAPITAL, POPULATION)');
IBQuery2.SQL.Add('VALUES (:Name, :Capital, :Population)');

IBQuery2.Params[0].AsString := 'Lichtenstein';
IBQuery2.Params[1].AsString := 'Vaduz';
IBQuery2.Params[2].AsInteger := 420000;
IBQuery2.ExecSQL;
```

## Prepared, Prepare Example

```
if not IBQuery1.Prepared then
begin
    IBQuery1.Close;
    IBQuery1.Prepare;
    IBQuery1.Open
end;
```



## SetData Example

```
var I: Longint;  
begin  
    I := Table1.FieldByName('CustID').AsInteger;  
    { Set the data }  
    Query1.ParamByName('CustNo').SetData(@I);  
end;
```

## UpdateRecordTypes, RevertRecord example

With minor coding, UpdateRecordTypes and RevertRecord can be used to undelete records when cached updates are enabled, as the following procedure demonstrates:

```
procedure UndeleteAll(DataSet: TIBCustomDataSet);  
begin  
  with DataSet do  
    begin  
      UpdateRecordTypes := [cusDeleted]; {make only deleted records visible}  
      try  
        First; {move to beginning of dataset}  
        while not EOF do  
          begin  
            RevertRecord; {undelete the current record}  
            Next; {move to the next record}  
          end;  
        UpdateRecordTypes := [cusUnInserted];  
        try  
          First; {move to beginning of dataset}  
          while not EOF do  
            begin  
              UnDeleteRecord; {undelete the current record}  
              Next; {move to the next record}  
            end;  
          finally  
            UpdateRecordTypes := [cusDeleted, cusModified, cusInserted,  
cusUnInserted, cusUnmodified];  
          end;  
        end;  
      end;  
    end;  
  end;
```



