

DataTable Custom Control (Version 3.0 - beta)

Copyright (C) - 1993
by
Douglas A. Bebbler
All rights reserved

Table of Contents

Introduction	3
DataTable Custom Control	5
DataTable / Paradox Engine Error Codes	45

Introduction

What is the DataTable?

The DataTable product is a Microsoft Windows Custom Control (DATATBL.VBX) designed to provide Visual Basic/Visual C++ programmers with a sophisticated, yet easy-to-use tool for building database management applications. Using the DataTable, Visual Basic programmers can build sophisticated multi-user, network compatible database management applications and distribute the DATATBL.VBX file with those applications on an unlimited, royalty-free basis. The DATATBL.VBX product presents the Visual Basic programmer with a simple, easy-to-use interface to Borland International's Paradox Engine. The Paradox Engine is a complete multi-user, network compatible API written in the C programming language. The DataTable product is a visual Custom Control interface to the Paradox Engine specifically designed for Visual Basic Programmers. DataTable (version 3.0) is compatible with Visual Basic 1.0 and 2.0 and runs in Microsoft Windows 3.0 as well as Microsoft Windows 3.1.

The DataTable (version 3.0) custom control product is shipped with two versions of the .VBX file:

- 1.) DATATBL.VBX (This is the DataTable custom control for application development.)
- 2.) DTBLRUN.VBX (This is the royalty-free runtime DataTable custom control to be distributed with your completed applications. Just rename the file to DATATBL.VBX with your distribution disks.)

This documentation describes the DataTable (3.0 -beta) product. It is a preliminary release of the DataTable (3.0) documentation set. This document is intended to provide a basic reference to the DataTable (3.0) custom control properties for developers until the official document set is released. The DataTable (3.0-beta) custom control package does not include a royalty-free runtime control, full documentation, or DataTable demos and sample applications. You may not distribute applications of any kind developed with the DataTable (3.0-beta) custom control unless you are specifically given permission to do so by the author. Violators will be prosecuted to the fullest extent provided by law.

Douglas A. Bebber
May 25, 1993

Trademarks

Visual Basic and Windows are registered trademarks of Microsoft Corporation.

Borland C++ is a registered trademark of Borland International.

PARADOX is a registered trademark of Borland International.

PARADOX Engine is a registered trademark of Borland International.

DataTable was written in Borland C++ (version 3.0) by Douglas A. Bebber. Address inquiries and bug reports (preferably Dr. Watson along with a listing of the suspected code) to

Douglas A. Bebber

Internet mail address:

bebberd@rr.bhc.edu

U.S. Postal Address:

1834 37th Street

Rock Island, Illinois 61201

Testing

This release of the DataTable custom control (version 3.0 - beta) is currently in beta test. If you discover any bugs or problems, I would appreciate a Dr. Watson UAE (General Protection Fault) report sent to my Internet address. Please describe your operating environment in detail and include a listing of your CONFIG.SYS and WIN.INI files.

Note: DataTable based Visual Basic programs will not be able to execute properly if the **PXENGWIN.DLL** file is not in a directory included in your MSDOS PATH statement. You must also have **SHARE.EXE** loaded to properly run the database engine environment. Version 3.0 - beta should have a Paradox Engine 3.X version of (**PXENGWIN.DLL**).

Note: DataTable will only execute in Windows Standard and 386 Enhanced modes.

Note: The DataTable (version 3.0 - beta) has support for Memo (M) BLOBs only. We are waiting for information from Borland International regarding the structured BLOB type (F,O,G) header formats. Also, as a part of this beta test distribution, we are asking developers for recommendations regarding the handling of Binary (B) BLOB types. How would you like to see the DataTable (version 3.0) handle multi-media objects such as .WAV files as well as bitmaps, icons, metafiles, .PCX, etc.

Compatibility and New Releases

The DataTable (version 3.0 - beta) is compatible with Borland International's Paradox Engine version 3.X. The 3.0 - beta release supports only Memo (M) BLOB types.

Disclaimer

This is a beta test release of the DataTable 3.0 custom control. Douglas A. Bebber makes no warranties as to the contents of this package and cannot be held responsible for any damages suffered from the use of this product.

DataTable Custom Control

DataTable

Description

DataTable objects provide Visual Basic programmers with the ability to interface with database files and the information present in them.

File Name

DATATBL.VBX

Object Type

DataTable

Remarks

The DataTable custom control is used to interface Visual Basic to database tables (files). Using this custom control Visual Basic programmers have access to multi-user, network compatible database resources.

The control has a few standard properties along with several DataTable specific properties. Since this control has been designed primarily to support Visual Basic programmers through code, there are no special "Visual" properties or elements in the DataTable custom control. As a matter of fact, the custom control itself is intended to be invisible at run-time.

You are able to interactively manipulate database tables at design time with the DataTable control. (You may wish to practice DataTable operations at design time to get familiar with the control.)

Before you are able to manipulate database tables those tables must already exist. The VBENGINE DATABASE TABLE MAKER utility program allows you to create PARADOX tables (those used with the DataTable custom control) interactively. **With the DataTable (version 3.0 - beta) custom control, you are now able to create Paradox tables (3.5 and 4.0 formats) in design mode as well as through code using the CreateTable Action.**

Distribution Note

When you create and distribute applications which use the DataTable custom control you should install the files **DTBLRUN.VBX (renamed as DATATBL.VBX)** and **PXENGWIN.DLL** in the customer's Microsoft Windows \SYSTEM subdirectory.

Properties

The properties for this control are listed below. Properties that apply only to this control are marked with an asterisk.

*Action	*IndexType	*RecordNumber	Top
*FieldBlank Visible	*KeySearch	*SaveEveryChange	
*FieldName	Left	*SearchMode	
*FieldNumber	*MemoOffset	*TableChanged	
*FieldType	*MemoSize	*TableFieldNames	
*FieldValue	Name	*TableFieldTypes	
*IndexCase	*NFields	*TableFound	
*IndexFieldName	*NRecords	*TableLockType	
*IndexFieldNames	*OtherTable	*TableName	
*IndexFile	*Password	*TableProtected	
*IndexID	*Reaction	*TableType	
*IndexNFields	*RecordLocked	Tag	

Action

Description

All database operations such as opening a database file, getting a record, getting a value from a field, etc. are classified as **Actions**. There are fifty-two DataTable Actions available for use in this version 3.0 - beta release of the DataTable.

Remarks

The Action property settings are:

Setting	Description

0	None (No action)
1	AddPassword
2	AddRecords
7	

3	AppendRecord
4	ClearRecord
5	CloseTable
6	CopyTable
7	CreateIndex
8	CreateTable
9	DecryptTable
10	DeleteIndex
11	DeleteRecord
12	DeleteTable
13	EncryptTable
14	FindTable
15	FirstRecord
16	FlushBuffers
17	GetField
18	GetFieldName
19	GetFieldNumber
20	GetFieldType
21	GetRecord
22	GetRecordNumber
23	GotoLock
24	GotoRecord
25	InsertRecord
26	IsFieldBlank
27	IsRecordLocked
28	IsTableProtected
29	LastRecord
30	LockRecord
31	LockTable
32	MapKey
33	NextRecord
34	NFields
35	NKeyFields
36	NRecords
37	OpenTable
38	PreviousRecord
39	PutBlank
40	PutField
41	QueryKey
42	RefreshTable
43	RemovePassword
44	RemoveRecords
45	RenameTable
46	SearchField
47	SearchKey
48	TableChanged
49	UnlockRecord
50	UnlockTable
51	UpdateRecord
52	UpgradeTable

Note: These Actions are defined in the **DATATBL.TXT** file as symbolic constants.

DataType

Integer (Enumerated)

DataTable Actions

0-None

Description

This Action does nothing.

1-AddPassword

Description

Enters a password into the system.

Remarks

This Action enters a password into the system. Up to 50 passwords can be concurrently active. No password may exceed 15 characters. The password to be added should be present in **DataTable.Password**. Upon a successful **AddPassword** Action, the **Reaction** property will contain a zero (0). In the event of an error, a non-zero error number is placed in the **Reaction** property.

See Also

RemovePassword, EncryptTable, DecryptTable, and IsTableProtected.

2-AddRecords

Description

Adds records from one table to another table.

Remarks

This Action adds records from the table specified in **DataTable.TableName** to the table specified in **DataTable.OtherTable**. If a key violation occurs, the record is overwritten. Both tables must exist and have compatible structures. Upon a successful Action the **Reaction** property will contain a zero (0). In the event of an error, a non-zero error number is placed in the **Reaction** property.

See Also

RemoveRecords.

3-AppendRecord

Description

Appends a record to a database table.

Remarks

This Action writes (appends) the record to the database file. If the database is indexed the **AppendRecord** Action works similar to the **InsertRecord** Action and the record is inserted into the database file at a place determined by the database index. If the database file is not indexed the appended record is added to the end of the database file. In both cases the newly appended record becomes the current record. Upon a successful Action the **Reaction** property will contain a zero (0). In the event of an error, a non-zero error number is placed in the **Reaction** property.

See Also

InsertRecord, UpdateRecord, DeleteRecord.

4-ClearRecord

Description

Clears out the current record for the specified database table.

Remarks

This function clears the DataTable's internal record information. Specifically all internal information for the DataTable's record structure is erased. It is a convenient way to clear all the field values for a specific record and is functionally equivalent to doing the **PutBlank** Action for each and every field. Upon a successful **ClearRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also **PutBlank**

5-CloseTable

Description

Closes a previously opened database table.

Remarks

This Action ensures that all buffered data is saved to disk and all memory allocated for the open table is released when the table is properly closed. When a Visual Basic

Program is finished with a database table it should do a **CloseTable** Action to insure that the table is properly closed and that no data is lost. Upon a successful **CloseTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

OpenTable

6-CopyTable

Description

Copies one table family to another.

Remarks

This Action copies a complete table (records included) and any family members to another identical table. The target destination table will be created if it does not exist. The table copied is the table specified in **DataTable.TableName** and the target destination table is specified in **DataTable.OtherTable**. Upon a successful **CopyTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

7-CreateIndex

Description

Creates an index for a database table.

Remarks

This Action creates an index (primary or secondary) on a table. The database table is specified in **DataTable.TableName**. The number of key fields in the index is specified in **DataTable.IndexNFields** (this should be 1 for all secondary indexes). The field number of the key field is specified in **DataTable.IndexID** (for primary indexes this should always be 1, for case sensitive, single field secondary indexes, IndexID should be set to the field number of the key field, and for case insensitive or composite secondary indexes you must obtain a field number via the **MapKey** Action.) The type of index to be created is specified in **DataTable.IndexType**. This property can have the following values:

0-Primary

1-NonMaintainedSecondary 2-MaintainedSecondary

Upon a successful **CreateIndex** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

DeleteIndex, MapKey and QueryKey

8-CreateTable

Description

Creates a database table.

Remarks

This Action creates a database table. The name of the new table is specified in **DataTable.TableName**. The type of table to be created (Paradox 3.5 or Paradox 4.0) is specified in **DataTable.TableType**. The number of fields that make up the new table's record structure is specified in **DataTable.NFields**.

The name of each field (each field separated by a comma) is specified in **DataTable.TableFieldNames**. The sequence of the field names placed in **DataTable.TableFieldNames** determines the sequence of the fields in the new table's record structure.

Each of the fields specified in **DataTable.TableFieldNames** must have a corresponding field type specified in **DataTable.TableFieldTypes**. Again each field type is separated by a comma. The ordering sequence of the field types should follow a one-to-one correspondence with the field names placed in **DataTable.TableFieldNames**.

Field names can each have a maximum of 25 characters. Field types 4 characters.

Each field name is separated from other field names by a comma. Note that the last field name in the list of field names should not have a comma after it.

Each field type is separated from other field types by a comma. Note that the last field type in the list of field types should not have a comma after it.

Upon a successful **CreateTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

CreateIndex, and DeleteTable.

9-DecryptTable

Description

Decrypts a database table.

Remarks

This Action decrypts a previously encrypted table. The target table is specified in **DataTable.TableName**. In order to successfully decrypt a table you must have previously entered the required password via a **AddPassword** Action. Upon a successful **DecryptTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

EncryptTable, AddPassword, and RemovePassword.

10-DeleteIndex

Description

Deletes a database index.

Remarks

This Action deletes either a primary or secondary index for the table specified in **DataTable.TableName**. The specific index to be deleted is specified in **DataTable.IndexID**. Upon a successful **DeleteIndex** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

CreateIndex, MapKey, and QueryKey.

11-DeleteRecord

Description

Deletes the current record from the database table.

Remarks

This Action deletes the current record in the database table. The database table and

the current record are contained inside the DataTable control. Upon a successful **DeleteRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

12-DeleteTable

Description

Deletes a database table.

Remarks

This Action deletes a database table (and any of the tables family objects). The database table to be deleted is specified in **DataTable.TableName** . Upon a successful **DeleteTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

DeleteIndex, and CreateTable.

13-EncryptTable

Description

Encrypts a database table.

Remarks

This Action encrypts a database table. The database table to be encrypted is specified in **DataTable.TableName** . In order to successfully encrypt a database table you must have previously entered a valid password into the system via a **AddPassword** Action. Upon a successful **EncryptTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

AddPassword, RemovePassword, and DecryptTable.

14-FindTable

Description

Checks to see if a table exists.

Remarks

This Action checks to see if the table specified in **DataTable.TableName** exists. If the table is found the **TableFound** property is set to 1-True. If the table was not found the **TableFound** property is set to 0-False. Upon a successful **FindTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

15-FirstRecord

Description

Positions the current record on the first record in the database table.

Remarks

This Action, if successful, moves to the first record in the database table and makes that record the current record. The database table and the current record for that table exist in the DataTable control.

Upon a successful **FirstRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

LastRecord, NextRecord and PreviousRecord.

16-FlushBuffers

Description

Saves all changes to disk.

Remarks

This Action, if successful, saves all changes to disk. This action is used when the **SaveEveryChange** property is set to 0-False. Upon a successful **FlushBuffers** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

SaveEveryChange property.

17-GetField

Description

Reads the value of a specified field from the current record of a database table.

Remarks

This Action reads the value of the field specified by **DataTable.FieldName** or **DataTable.FieldNumber** and places that field's value in **DataTable.FieldValue**. The field value read is that of the current record in the database table. All field values placed in **DataTable.FieldValue** are of type string regardless of the actual data type stored in the table itself. Upon a successful **GetField** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

GetFieldName, GetFieldNumber, GetFieldType, and PutField.

18-GetFieldName

Description

Reads the name of a database field.

Remarks

This Action reads the value of the field specified by **DataTable.FieldNumber** and places that field's name in **DataTable.FieldName**. Upon a successful **GetFieldName** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

GetField, GetFieldNumber, GetFieldType, and PutField.

19-GetFieldNumber

Description

Reads the field number of a database field.

Remarks

This Action reads the number of the field specified by **DataTable.FieldName** and places that field's number in **DataTable.FieldNumber**. Upon a successful **GetFieldNumber** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

GetField, GetFieldName, GetFieldType, and PutField.

20-GetFieldType

Description

Gets the data type for a database field.

Remarks

This Action returns the data type of the field specified in **DataTable.FieldName**. You use this function when you wish to determine the actual data type of the field as it is stored in the database table. The possible data types returned are as follows:

Field Type	Data Type
N	Numeric
S	Short number
\$	Currency
Annn	Alphanumeric
D	Date
M	Memo BLOB
F	Formatted memo BLOB
B	Binary BLOB
G	Graphic BLOB
O	OLE BLOB

Note: The DataTable (version 3.0 - beta) custom control supports only the Memo (**M**) BLOB type.

The field type is returned in the **DataTable.FieldType** property. Upon a successful **GetFieldType** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

21-GetRecord

Description

Reads the current record in the database table.

Remarks

This Action, if successful, reads the current record in the database table. The database table and the current record for that table are present in the DataTable control. Upon a successful **GetRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

FirstRecord, LastRecord, NextRecord and PreviousRecord.

22-GetRecordNumber

Description

Gets the database record number of the current record.

Syntax

Remarks

The **GetRecordNumber** Action gets the record number of the current record. The current record and database table are present in the DataTable control. The record number is placed in the **DataTable.RecordNumber** property. Upon a successful **GetRecordNumber** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

23-GotoLock

Description

Moves to a previously locked record.

Remarks

This Action, if successful, moves to the previously locked record. Upon a successful

GotoLock Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

LockRecord.

24-GotoRecord

Description

Goes to the specified record number in the database table and makes that record the current record.

Remarks

This Action moves to the **DataTable.RecordNumber** record in the database table and makes that record the current record. Upon a successful **GotoRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property

25-InsertRecord

Description

Inserts a record into the database table file.

Remarks

This Action inserts a record into the database table file. If the database file is indexed the **InsertRecord** Action works similar to the **AppendRecord** Action and the record is inserted in the database file at a location specified by the database index. If the database file is not indexed the new record is inserted before the current record. In both cases the newly inserted record becomes the current record. Upon a successful **InsertRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

AppendRecord, UpdateRecord, DeleteRecord.

26-IsFieldBlank

Description

Tests to see if a field is blank.

Remarks

This Action tests to see if the field specified in **DataTable.FieldNumber** is blank. If the field value is blank the **FieldBlank** property is set to 1-True. If the field value is not blank, the **FieldBlank** property is set to 0-False. Upon a successful **IsFieldBlank** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

PutBlank and the FieldBlank property.

27-IsRecordLocked

Description

Tests to see if the current database record is locked.

Remarks

This Action tests to see if the field specified in **DataTable.FieldNumber** is blank. If the current record is locked the **RecordLocked** property is set to 1-True. If the current record is not locked, the **RecordLocked** property is set to 0-False. Upon a successful **IsRecordLocked** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

LockRecord and UnlockRecord.

28-IsTableProtected

Description

Tests to see if a table is encrypted.

Remarks

This Action tests to see if a table is encrypted. If the table specified in **DataTable.TableName** is encrypted, the **TableProtected** property is set to 1-True. Otherwise the **TableProtected** property is set to 0-False. Upon a successful **IsTableProtected** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

EncryptTable and DecryptTable.

29-LastRecord

Description

Moves to the last record in the database table.

Remarks

This Action moves to the last record in the database table and makes that record the current record. Upon a successful **LastRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

FirstRecord, NextRecord and PreviousRecord

30-LockRecord

Description

Locks the current database record.

Remarks

This Action locks the current record. The database table and its current record are specified in the DataTable control. Once the record is successfully locked, no other users are able to delete, or otherwise write to the record until the record is unlocked with an **UnlockRecord** Action. Upon a successful **LockRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

UnlockRecord

31-LockTable

Description

Locks a database table.

Remarks

This Action locks the database table specified in **DataTable.TableName**. The type of lock applied to the table is specified in **DataTable.TableLockType**. Upon a successful **LockTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

UnlockTable and **TableLockType** property.

32-MapKey

Description

Obtains an **IndexID** for a composite or case-insensitive, single field index.

Remarks

This Action maps an ordered set of fields or a case-insensitive single field index to a special **IndexID** which can be used as an index in key search operations. The **MapKey** Action expects certain information to be present in specific DataTable properties:

DataTable.TableName specifies the target database table.

DataTable.IndexNFields specifies the number of key fields composing the index.

DataTable.IndexFieldNames specifies the individual fields composing the index (fields are separated from one another by commas.)

DataTable.IndexFieldName specifies the name of the new key field. (the name you wish to use to reference this key field by in subsequent database operations - 25 characters maximum.)

DataTable.IndexCase specifies whether the new index will be a case-sensitive or case-insensitive index.

After a successful **MapKey** Action, the **IndexID** property will have a value (>255) which will be used to reference this special key in subsequent database operations.

Upon a successful **MapKey** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

CreateIndex, and **QueryKey**.

33-NextRecord

Description

Moves to the next record in the database table.

Remarks

This Action moves to the next record in the database table and makes that record the current record. The database table is specified in the DataTable control. Upon a successful **NextRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

FirstRecord, LastRecord, and PreviousRecord.

34-NFields

Description

Gets the number of fields in the database record structure.

Remarks

This Action gets the total number of fields present in the database table's record structure. The database table is specified in **DataTable.TableName**. The number of fields present in the record structure is placed in the **DataTable.NFields** property. Upon a successful **NFields** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

NKeyFields.

35-NKeyFields

Description

Gets the number of key fields in the database record structure.

Remarks

This Action gets the total number of key fields present in the database table's record structure. The database table is specified in **DataTable.TableName**. The number of key fields present in the record structure is placed in the **DataTable.NKeyFields** property. Upon a successful **NKeyFields** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

NFields.

36-NRecords

Description

Gets the number of records present in the database table.

Remarks

This Action gets the total number of records present in the database table specified in the DataTable control. The number of records is placed in the **DataTable.NRecords** property. Upon a successful **NRecords** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

37-OpenTable

Description

Opens a database table file for subsequent processing.

Remarks

Before you can process information in a database table file, you must first open that file for processing. You open database table files by performing the **OpenTable** Action. To successfully open a database table you will need to specify three other DataTable properties:

DataTable.TableName
DataTable.IndexID
DataTable.SaveEveryChange

DataTable.TableName should hold the name of the database table file including any MSDOS PATH specifier. Do not include the file extension.

DataTable.IndexID should specify the index you wish to use for table operations. **MASTERINDEX** should be used to open the table with all of its associated indexes. For a specific index, specify the field number of the associated index.

DataTable.SaveEveryChange should specify whether you wish to save every change to disk or whether you wish to buffer changes to disk. Buffering is faster, but you may lose data if the power goes out (see **FlushBuffers** for information on writing buffered data to disk). To buffer changes set this parameter to FALSE.

Once these three DataTable properties have been appropriately set, perform an **OpenTable** Action to open the database table. Upon a successful **OpenTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

CloseTable, FlushBuffers.

38-PreviousRecord

Description

Moves to the previous record in the database table.

Remarks

This Action moves to the previous record in the database table and makes that record the current record. The database table is specified in the DataTable control. Upon a successful **PreviousRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

FirstRecord, LastRecord, and NextRecord.

39-PutBlank

Description

Places a blank value into the specified field in the database record.

Remarks

This Action places a blank value into the field specified in the **DataTable.FieldName** property. The field value is not written to the database table until the record is written to disk using either **InsertRecord**, **AppendRecord**, or **UpdateRecord** Actions. A blank value of the appropriate data type is placed in the field automatically. A blank value is a valid value which represents the fact that the value has yet to be entered (a blank value is not zero.) Upon a successful **PutBlank** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

40-PutField

Description

Places a field value into the specified field in the database record.

Remarks

This Action places the value found in **DataTable.FieldValue** for the field **DataTable.FieldName** into the database record. The record in the database table file is not actually modified until an **InsertRecord**, **AppendRecord**, or **UpdateRecord** Action is performed. The table and record for the operation is specified by the DataTable control. All field values to be written to a database field are placed in **DataTable.FieldValue** and are of type String regardless of the actual data type of the field in the database table itself. The **PutField** Action automatically converts the value to the appropriate type before placing it in the database record. Upon a successful **PutField** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

GetField, **PutBlank**.

41-QueryKey

Description

Gives information about a database index..

Remarks

This Action places information concerning the index file specified in **DataTable.IndexFile** into several of the DataTables properties:

DataTable.IndexFieldName	specifies the field name of the index key field.
DataTable.IndexNFields	specifies the number of key fields composing the index.
DataTable.IndexCase	specifies whether the index is a case-sensitive or case-insensitive index.
DataTable.IndexFieldNames	specifies the key field of the index (field names separated by a comma.)
DataTable.IndexID	specifies the I.D. for the index.

Upon a successful **QueryKey** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

MapKey.

42-RefreshTable

Description

Refreshes or updates a table image to reveal up-to-the minute changes.

Remarks

This Action updates the table image to reflect any changes to data that other users may have made since your last table refresh. The following Actions automatically refresh a table image **RecordLock, UpdateRecord, InsertRecord, AppendRecord, and DeleteRecord**. Upon a successful **RefreshTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

43-RemovePassword

Description

Removes a previously entered password from the system.

Remarks

This Action removes a previously entered password (entered via **AddPassword** Action) from the system. The password to be removed should be present in **DataTable.Password**. Upon a successful **RemovePassword** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

AddPassword, EncryptTable, DecryptTable and IsTableProtected.

44-RemoveRecords

Description

Removes all the records from a database table.

Remarks

This Action removes all records from the database table specified in **DataTable.TableName**. Upon a successful **RemoveRecords** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

45-RenameTable

Description

Renames a database table.

Remarks

This Action renames a database table (and table family members if any.) The table is specified in **DataTable.TableName**. The new name for the table is specified in **DataTable.OtherTable**. Upon a successful **RenameTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

46-SearchField

Description

Searches a database table file on a specified field.

Remarks

This Action searches through the database table for a value in a field. The database field searched on is specified by **DataTable.FieldName** the field value to search for is specified by **DataTable.FieldValue**. You need to set these two properties and then perform the **PutField** Action. After that you need to specify your search mode preference by setting **DataTable.SearchMode** to one of three values:

- **SEARCHFIRST**
- **SEARCHNEXT**
- **CLOSESTRECORD**

SEARCHFIRST begins the search at the first record in the database, the record position of the current record is not changed if a search attempt fails to find a match.

SEARCHNEXT begins with the record following the current record in the database, the record position of the current record is not changed if a search attempt fails to find a match.

CLOSESTRECORD begins to search at the first record in the database, if a record is not found (search attempt fails), one of two possibilities exist:

-If there is no exact match, there happens to be a record which has a value lexically greater than the search value. The current record in the database will be the record with the first such instance and a record not found error (89) returned.

- There is no record in the database that has a value greater or equal to the search value. The current record will be the last record in the database and a record not found error (89) returned.

A search can then be started with a call to the **SearchField** function.

The available search modes rely on the index on which the table is currently using. **SearchField** always searches for the first record which fullfills the search criteria. On non-indexed database tables **SearchField** searches via a sequential scan. The order of the records searched through the sequential scan is that of the physical order of the records in the table itself. In non-indexed tables **CLOSESTRECORD** is not supported. Upon a successfull **SearchField** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

SearchKey

47-SearchKey

Description

Searches a database table for a key match.

Remarks

This Action searches the table specified in **DataTable.TableName** on the Primary index. A search match is sought on the key field(s) of the table specified by **DataTable.SearchKey**. The key to be matched must be the primary key or a subset of the primary key. The fields to be matched are the fields which have been placed into the database engine's record buffer via **PutField** Actions.

If there are five key fields and you are only interested in finding records which have specific values in the first two key fields lets say "Date" and "Customer Name", you want to search for records in the database that have 12/12/92 for the "Date" value and "Robert Smith" for the "Customer Name" you would set the criteria for those fields and place them in the database engine via **PutField** Actions. Your KeySearch would be set up as **DataTable.KeySearch = 2**.

You need to specify your search mode preference by setting **DataTable.SearchMode** to one of three values:

- **SEARCHFIRST**
- **SEARCHNEXT**
- **CLOSESTRECORD**

SEARCHFIRST begins the search at the first record in the database, the record position of the current record is not changed if a search attempt fails to find a match.

SEARCHNEXT begins with the record following the current record in the database, the record position of the current record is not changed if a search attempt fails to find a match.

CLOSESTRECORD begins to search at the first record in the database, if a record is not found (search attempt fails), one of two possibilities exist:

-If there is no exact match, there happens to be a record which has a value lexically greater than the search value. The current record in the database will be the record with the first such instance and a record not found error (89) returned.

- There is no record in the database that has a value greater or equal to the search value. The current record will be the last record in the database and a record not found error (101) returned.

The available search modes rely on the index on which the table is currently using. **SearchKey** always searches for the first record that fullfills the search criteria. Once the desired key fields have been set-up and submitted via **PutField** Actions, the desired search mode specified, along with the keysearch specification, you can then do a **SearchKey** Action. Upon a successfull **SearchKey** Action, an integer value of zero

(0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

SearchField.

48-TableChanged

Description

Indicates if table image has been changed by other users.

Remarks

This Action indicates whether a database table image has been significantly changed by other users (enough to warrant a **RefreshTable** Action.) The table is specified in **DataTable.TableName**. If the table image has been changed the **DataTable.TableChanged** property is set to 1-True, otherwise it is set to 0-False.

Upon a successful **TableChanged** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

RefreshTable

49-UnlockRecord

Description

Unlocks a previously locked record.

Remarks

This Action unlocks a previously locked record. You are only able to unlock records that you have previously locked. You can not unlock records locked by other users. A locked record can also be unlocked under the following conditions:

- You delete the record by performing a **DeleteRecord** Action.
- You do a **CloseTable** Action which unlocks all the records in that table before closing the table.

Upon a successful **UnlockRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

LockRecord

50-UnlockTable

Description

Unlocks a previously locked database table.

Remarks

This Action unlocks a previously locked database table. The table is specified in **DataTable.TableName**. The table lock type must be specified in **DataTable.TableLockType**.

Upon a successful **UnlockTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

LockTable

51-UpdateRecord

Description

Updates a record in a database table.

Remarks

This Action updates the record specified in the DataTable control to the database file. There must be a current database record to update. Upon a successful **UpdateRecord** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

See Also

AppendRecord, InsertRecord, DeleteRecord.

52-UpgradeTable

Description

Upgrades an older Paradox table (3.5 or later) to Paradox 4.0 table format.

Remarks

This Action upgrades an older Paradox table to the new Paradox 4.0 table format. Upon a successful **UpgradeTable** Action, an integer value of zero (0) is placed in the DataTable's **Reaction** property. In the event of an error, a non-zero integer error value is placed in the **Reaction** property.

FieldBlank

Description

This property is an integer value which represents whether or not a database field is blank.

This property is not automatically managed for reasons of performance. You should only consider this property's value valid directly after performing a **FieldBlank** Action.

This property may have the following values:

Setting	Description
0	False
1	True

DataType

Integer (Enumerated)

FieldName

Description

This property is an ASCII string with a maximum length of 25 characters. This string holds the name of the target database field.

DataType

String

33

FieldNumber

Description

This property is an integer value which represents the field's position in the table's record structure.

This property is not automatically managed for reasons of performance. You should only consider this property's value valid directly after entering it manually or after performing a **FieldNumber** Action.

DataType

Integer

FieldType

Description

This property is an ASCII string with a maximum length of 5 characters. This property holds the data type of the target database field.

Database Field Types

Each field in a database has a corresponding data type. The available field types in the DataTable (version 1.0) release are listed below:

Alphanumeric (A) field type permits the full ASCII character set (except ASCII 0) and is used for entry of string data types. Fields of this type are specified as **Axxx**, where the **xxx** represents the maximum length of the field in characters. For example, if you were to create a field in a table which is intended to hold a maximum of 50 characters you would specify the field as an **A50**.

Number (N) and **currency (\$)** field types permit up to 15 significant digits (including the decimal point) in the range of real numbers from $\pm 10^{-307}$ to $\pm 10^{307}$. Number field values which are greater than 15 significant digits are rounded and stored in scientific notation. Currency field values are stored in a default predefined format.

Short Number (S) field types permit values in the range of signed integers. (-32,767 to 32,767).

Date (D) field types permit any valid dates between January 1, 100 A.D. to December 31, 9999. Date values are stored as long integers which represent the number of days since January 1, A.D.

Memo (M) field type contains text that is variable in length and usually larger than 255 characters. Fields of this type can be specified as **Mxxx**, where the **xxx** represents how many characters are

actually stored in the database table. The entire memo is also stored in a .MB file which has the same filename as the table.

BLOB types (F,O,G and B) are not supported in release 3.0 - beta.

DataType

String

FieldValue

Description

This property is an ASCII string with a maximum length of 255 characters. This property holds the value of the target database field.

Remarks

DataTable programming involves handling database field values as strings only! Regardless of the actual data type in the database file. This is mandated by the DataTable's internal data structures. DataTable programmers receive field values from data table files as String values and write database field values as String values regardless of the actual field value type present in the database table file. The DataTable automatically performs data type conversions based on the data type of the field in the database table file. This data type conversion process is transparent to the Visual Basic programmer and provides a much simpler interface to database programming.

DataType

String

IndexCase

Description

This property is an integer (enumerated) which is used to specify whether an index is case-sensitive or case-insensitive.

This property may have the following values:

Setting	Description
0	False
1	True

DataType

Integer (Enumerated)

IndexFieldName**Description**

This property is an ASCII string which is used to hold the name of a programmer supplied index field name. This property is used strictly for the **MapKey** and **QueryKey** Actions.

DataType

String

IndexFieldNames**Description**

This property is an ASCII string which is used to hold the names of key fields composing an index. Individual field names must be separated by a comma. The last field name in this list must **not** be followed by a comma.

DataType

String

IndexFile**Description**

This property is an ASCII string which is used to hold the name of an index file (a path specification may be included) for the **QueryKey** Action.

DataType

String

IndexID**Description**

This property is an integer which holds the identification of the index to be used with the database table (specified in the **DataTable.TableName** property).

Remarks

This property should be zero (0) to open the database table with all associated indexes.

DataType

Integer

IndexNFields

Description

This property is an integer which holds the number of key fields present in an index. This property is not automatically maintained. It is used in conjunction with the **MapKey**, **QueryKey**, and **CreateIndex** Actions.

DataType

Integer

IndexType

Description

This property is an integer (enumerated) which is used to specify the type of index.

This property may have the following values:

Setting	Description
0	Primary
1	NonMaintainedSecondary
2	MaintainedSecondary

DataType

Integer (Enumerated)

KeySearch

Description

This property is an integer data member which specifies what portion of the databases primary index to use for index based searches.

Remarks

37

All database **key** fields must be contiguous fields starting with the first field in the database table. All database searches performed with the **SearchKey** Action must specify the portion of the **PRIMARY** index to search on. For example:

If there are five key fields in your database table and you are only interested in finding records which have specific values in the first two key fields lets say "Date" and "Customer Name", you want to search for records in the database that have 12/12/92 for the "Date" value and "Robert Smith" for the "Customer Name" you would set the criteria for those fields and place them in the database engine via **PutField** Actions. Your KeySearch would be set up as **DataTable.KeySearch = 2**.

An example of how to use **key** fields and database searches using the **SearchKey** Action is provided in the **DataTable Example Programs** section of this document. (Extensive examples of using **key** fields and database searches using full and partial keys are provided in the DataTable User's Manual.)

DataType

Integer

MemoOffset

Description

This property is a long value which is used to mark an offset into a memo field.

Remarks

This property is used when reading and writing memo fields which are larger than 64 Kilo bytes in length. Memo fields are sourced (read/written) from a Visual Basic String (HSZ). Visual Basic strings have a maximum length of 64 KB. When reading/writing memo fields larger than 64 KB you must do so in chunks of 64 KB.

The **MemoOffset** property is used in conjunction with the **MemoSize** property. When reading in any Memo field the **MemoSize** property is automatically updated by the DataTable custom control. The value present in the **MemoSize** property will tell you the total length (Size) of the Memo field (in bytes). If the size indicated by **MemoSize** is greater than 64 KB, you will need to read in multiple 64 KB chunks of data. The **MemoOffset** property provides a means of indexing in the Memo field relative to **MemoSize**.

DataType

long

MemoSize

Description

38

This property is a long value which is used to determine the total length (in bytes) of a stored Memo field value.

Remarks

This property is used when reading memo fields. When reading any Memo field the **MemoSize** property is automatically updated to indicate the total size of the stored Memo (in bytes). When reading in Memo fields which are larger than 64 KB (65, 520 bytes to be exact), you must do so in chunks of approximately 64 KB. This limitation is imposed on two sides.

1. The Paradox Engine 3.0 can only read/write BLOBs in chunks of 65, 520 bytes. For BLOBs larger than 65,520 bytes, you must perform multiple read/write operations.

2. The maximum size of a Visual Basic String data type is approximately 64 KB (overhead is involved). Currently DataTable Memo fields are read/written through Visual Basic strings (HSZ for Visual C++ users).

All Memo based read/write operations must be performed within the confines of the above two imposed limitations.

Data Type

long

NFields

Description

This property is an integer value which indicates the number of fields present in the database table's record structure.

Remarks

This property is not automatically managed by the DataTable control due to performance reasons. Therefore to insure that the value present in this property is as accurate as possible read the value immediately after performing a **NFields** Action on the table.

Data Type

Integer

NKeyFields

Description

This property is an integer value which indicates the number of key fields present in the database table's record structure.

Remarks

39

This property is not automatically managed by the DataTable control due to performance reasons. Therefore to insure that the value present in this property is as accurate as possible read the value immediately after performing a **NKeyFields** Action on the table.

DataType

Integer

NRecords

Description

This property is a long value which indicates the number of records present in the database table.

Remarks

This property is not automatically managed by the DataTable control due to performance reasons and complications related to network concurrency. Therefore to insure that the value present in this property is as accurate as possible read the value immediately after performing a **NRecords** Action on the table.

DataType

long

OtherTable

Description

This property is an ASCII string used to hold the name of a database table (path may be specified.) This property is used in DataTable actions that reference another table such as **AddRecords**, **RenameTable**, **CopyTable**, etc.

DataType

string

Password

Description

This property is an ASCII string used to hold a password for DataTable Actions such as **AddPassword** and **RemovePassword**. Passwords may have no more than 15 characters.

DataType

string

40

Reaction

Description

This property is an integer value which indicates whether or not a DataTable Action was performed successfully or not.

Remarks

Its purpose is to report any errors encountered when performing any DataTable Actions. For a listing of the possible errors reported after performing DataTable Actions see the DataTable / PARADOX ENGINE ERROR CODES section at the end of this document.

DataType

Integer (Enumerated)

RecordLocked

Description

This property is an integer (enumerated) which is used to indicate whether or not the current record is locked.

This property is not automatically managed, so it should only be considered accurate directly after performing a **IsRecordLocked** Action.

This property may have the following values:

Setting	Description
0	False
1	True

DataType

Integer (Enumerated)

RecordNumber

Description

This property is a long value which indicates the record number of the current record in the

database table.

Remarks

This property is not automatically managed by the DataTable control due to performance reasons and complications related to network concurrency. Therefore to insure that the value present in this property is as accurate as possible read the value immediately after performing a **GetRecordNumber** Action on the table.

DataType

long

SaveEveryChange

Description

This property is an integer value which indicates whether changes to database tables are done immediately or buffered to disk.

Remarks

This property is available to programmers basically for performance reasons. The possible values of this property are:

Setting	Description
0	False
1	True

DataType

Integer (Enumerated)

SearchMode

Description

This property is an integer value which indicates the search mode used in database searches.

Remarks

The possible values of this property are:

Setting	Description
---------	-------------

0	SEARCHFIRST
1	SEARCHNEXT
2	CLOSESTRECORD

SEARCHFIRST begins the search at the first record in the database, the record position of the current record is not changed if a search attempt fails to find a match.

SEARCHNEXT begins with the record following the current record in the database, the record position of the current record is not changed if a search attempt fails to find a match.

CLOSESTRECORD begins to search at the first record in the database, if a record is not found (search attempt fails), one of two possibilities exist:

-If there is no exact match, there happens to be a record which has a value lexically greater than the search value. The current record in the database will be the record with the first such instance and a record not found error (89) returned.

- There is no record in the database that has a value greater or equal to the search value. The current record will be the last record in the database and a record not found error (101) returned.

Data Type

Integer (Enumerated)

TableChanged

Description

This property is an integer (enumerated) which is used to indicate whether or not the current table image is still valid.

This property is not automatically managed, so it should only be considered accurate directly after performing a **TableChanged** Action.

This property may have the following values:

Setting	Description
0	False
1	True

Data Type

Integer (Enumerated)

TableFieldNames

Description

This property is an ASCII string which is used to hold the names of fields in a database table. Individual field names must be separated by a comma. The last field name in this list must **not** be followed by a comma.

This property is used primarily with the **CreateTable** Action.

DataType

String

TableFieldTypes

Description

This property is an ASCII string which is used to hold the field types of the fields listed in the **TableFieldNames** property. The field types listed must be on a one-to-one correspondence with the field names in the **TableFieldNames** property. Individual field names must be separated by a comma. The last field name in this list must **not** be followed by a comma.

This property is used primarily with the **CreateTable** Action.

DataType

String

TableFound

Description

This property is an integer (enumerated) which is used to indicate whether or not a table exists. It is used only with the **FindTable** Action.

This property is not automatically managed, so it should only be considered accurate directly after performing a **FindTable** Action.

This property may have the following values:

Setting	Description
0	False
1	True

DataType

Integer (Enumerated)

TableLockType

Description

This property is an integer (enumerated) which is used to specify the table lock type for the **LockTable** and **UnlockTable** Actions.

This property may have the following values:

Setting	Description
0	None
1	FullLock
2	WriteLock
3	PreventWriteLock
4	PreventFullLock

DataType

Integer (Enumerated)

TableName

Description

This property is an ASCII string with a maximum length of 255 characters. This string holds the name of a database table, including any MSDOS PATH specifier. Database file names placed in this property must not include a file extension.

DataType

String

TableProtected

Description

This property is an integer (enumerated) which is used to indicate whether or not a table is encrypted. It is used only with the **IsTableProtected** Action.

This property is not automatically managed, so it should only be considered accurate directly after

performing a **IsTableProtected** Action.

This property may have the following values:

Setting	Description
0	False
1	True

DataType

Integer (Enumerated)

TableType

Description

This property is an integer (enumerated) which is used to indicate the type of table (Paradox 3.5 or 4.0.) It is used only with the CreateTable Action.

The default is Paradox 4.0

This property may have the following values:

Setting	Description
0	Paradox35
1	Paradox40

DataType

Integer (Enumerated)

DataTable / PARADOX ENGINE ERROR CODES

Error Code	Description
46	

1	Drive not ready
2	Directory not found
3	File is busy
4	File is locked
5	File not found
6	Table damaged
7	Primary index damaged
8	Primary index is out of date
9	Record is locked
10	Sharing violation - directory busy
11	Sharing violation - directory locked
12	No access to directory
13	Sort for index different from table
14	Single user but directory is shared
15	Multiple PARADOX.NET files found
21	Insufficient password rights
22	Table is write-protected
30	Data type mismatch
31	Argument is out of range
33	Invalid argument
40	Not enough memory to complete operation
41	Not enough disk space to complete operation
50	Another user deleted record
51	BLOB open mode, action N/A
52	BLOB already open
53	Invalid BLOB offset
54	Invalid BLOB size
55	Other user modified BLOB
56	Bad BLOB file
57	Can not index on BLOB
59	Bad BLOB handle
60	Can not search on BLOB
70	No more file handles available
72	No more table handles available
73	Invalid date given
74	Invalid field name
75	Invalid field handle
76	Invalid table handle
78	Engine not initialized
79	Previous fatal error, cannot proceed
81	Table structures are different
82	Engine already initialized
83	Unable to perform operation on open table
86	No more temporary names available
89	Record was not found
94	Table is indexed
95	Table is not indexed
96	Secondary index is out of date
97	Key violation
98	Could not login on network
99	Invalid table name
101	End of table
102	Start of table
103	No more record handles available
104	Invalid record handle
105	Operation on empty table
106	Invalid lock code
107	Engine not initialized
108	Invalid file name
109	Invalid lock
110	Invalid lock handle
111	Too many locks on table
112	Invalid sort-order table
113	Invalid net type
114	Invalid directory name
115	Too many passwords specified
116	Invalid password

117	Buffer too small for result
118	Table is busy
119	Table is locked
120	Table was not found
121	Secondary index was not found
122	Secondary index is damaged
123	Secondary index is already open
124	Disk is write-protected
125	Record is too big for index
126	General hardware error
127	Not enough stack space to complete operation
128	Table is full
129	Not enough swap buffer space to complete operation
130	Table is SQL replica
131	Too many clients for Engine DLL
132	Exceeds limits specified in WIN.INI
133	Too many files open simultaneously (includes all clients)
134	Can't lock PARADOX.NET - is SHARE.EXE loaded
135	Can't run Engine in Windows real mode
136	Can't modify unkeyed table with non-maintained secondary index
137	Timed out performing lock operation.