

DataTable Custom Control (Version1.1)
Working Model

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

Table of Contents

Introduction	3
Database Fundamentals	6
DataTable Custom Control	11
DataTable Example Programs	16
DataTable / Paradox Engine Error Codes	57

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 1.1) 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 1.1) working model product itself contains a rather extensive subset of the full-featured DataTable product (available only to registered users). Even though it is a subset, it has all the essential functionality necessary to design full-featured database applications. As a matter of fact, you can design a very sophisticated package in its entirety using only the working model. I included this functionality in the working model so you could better evaluate the product and its capabilities before registering. However, you are not allowed to distribute applications designed around the working model. You must first obtain a registered copy of the DataTable product before you are allowed to distribute the DataTable with your commercial applications (this includes Shareware products).

The DataTable working model has been called a "NagWare" product. There is a pop-up window in the working model that constantly reminds users that the product is for pre-registration evaluation only and cannot be distributed as part of any product. It is displayed throughout portions of the calling programs code. When you register your copy of the DataTable, you get two versions of the DATATBL.VBX (one for development and another for product distribution) both without the "NagWare" pop-up window.

This documentation describes the DataTable product. It specifically describes the DataTable (version 1.1) working model which is currently being distributed over computer bulletin board services in the United States of America. The DataTable (version 1.1) working model is a Shareware product, it is copyrighted and I reserve all rights to it. You may distribute it to others, through any means, as long as you do not charge others for the product itself, or alter the product in any way.

Douglas A. Bebbber
May 6, 1993

How to Register

You can obtain a registered copy of the DataTable (version 1.1) product for only \$49.95. The package includes:

- A full-featured DATATBL.VBX (access to all functions\properties minus the "NagWare" pop-up). (one .VBX for development and another for product distribution.)
- DataTable users manual complete with example Visual Basic programs and source code.
- DataTable Technical Reference Manual describing all DataTable properties in detail.
- Unlimited, royalty-free rights to distribute the run-time DATATBL.VBX with your applications.
- Notice of product updates.
- Free telephone technical support.

To register send check or money order to:

**Douglas A. Bebber
1834 37th Street
Rock Island, Illinois 61201
(309) 786-9602**

(make notes payable to: Douglas A. Bebber)

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. Bebbber. Address inquiries and bug reports (preferably Dr. Watson along with a listing of the suspected code) to

Douglas A. Bebbber

Internet mail address:
bebbberd@rr.bhc.edu

U.S. Postal Address:
1834 37th Street
Rock Island, Illinois 61201

Testing

DataTable was written and tested on a variety of 286, 386, and 486 PCs. Record and file locking functions were tested and verified on Lantastic and Novell based ethernet LANs as well as in the standard Windows environment between multiple applications.

If your LAN hardware or software differs significantly and DataTable does not run properly, 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 **DATATBL.VBX** and **PXENGWIN.DLL** files are not in directories included in your MSDOS PATH statement. You must also have **SHARE.EXE** loaded to properly run the database engine environment.

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

Compatibility and New Releases

The DataTable (version 1.1) is compatible with Borland International's Paradox Engine version 2.0. Registered users of the VBENGINE 1.0 product will receive a **free** upgrade to the DataTable (version 1.1) product.

Additional functions/features present in the registered version of the DataTable product:

- + File lock/unlock capabilities.
- + File encrypt/decrypt capabilities.
- + Table create/delete routines (via code)
- + Index create/delete routines (PRIMARY and SECONDARY indexes via code)
- + Network user identification routines
- + Table copy/add/rename/empty and other similar routines.

Database Fundamentals

What is a Database?

For our purposes we will limit this discussion to the world of IBM PC compatible database management systems. Specifically, relational database management systems designed around Borland International's Paradox Engine and the user friendly Visual Basic custom control (the DataTable).

In this context, we can say that a database consists of one or more related files (tables in DataTable terminology) that hold information in an orderly, efficient manner. The database tables consist of several rows and columns into which, information is placed. The columns are generally referred to as "Fields" and the rows as "Records".

We will not delve into a lot of theoretical concepts in this section, rather we will present concepts in order to promote a general understanding of database principles. Just enough to give the beginner a kick-start into the world of database programming. (the DataTable User's manual covers a little more theory and provides references to more extensive texts.)

To illustrate the principles involved in simple database design we will present a model which we will build on in the DataTable Programming Examples section of this manual. To start, we will design a simple database which will hold information concerning the customers of a small business owner. We will design the database from scratch and will detail its structure in this section.

Our small business owner tells us that he would like to maintain a certain set of facts concerning each one of his customers. Specifically, he would like to have the following information concerning each customer in his database:

- 1.) The customer's Name**
- 2.) The customer's street address**
- 3.) The customer's city of residence**
- 4.) The state that the customer lives in**
- 5.) The customers zip code**
- 6.) The customer's telephone number**

This listing of required information is the first essential step in the design of database systems. It is absolutely essential that you compile the needed sets of information that must be maintained. In the world of database programming this step is called "compiling a data dictionary". Over time additional items of information get added to the list. Sometimes, certain items need to be broken down into several smaller parts so a more detailed or "higher resolution" picture can emerge.

The items listed above are the essential pieces of information our small business owner requires for each of his customers. Each piece of information is needed for every customer. If we think of this conceptually, the required information expands in only one direction. Every time we compile the required set of information for a customer our information grows. Its not that we get more information or details concerning the customer, but that we get more customers with the same set of information (Name, Address, etc.).

In a computer-based database system we generally define a set of data that we would like to obtain for each new entry into the database system. This set of information is called the database field set, consisting of a finite set of fields. The six items listed above are our database fields. The entries in our database, each consisting of the same set of fields, are our records. Each and every one of our customers constitute an individual record in our data base. As you can see, our set of

required fields should remain more or less constant. But we hope that our customer base continues to expand. In general databases expand in one direction, in our terms, horizontal, record expansion.

Given the above, we can now look at the structure of our database diagramed below (fields vertically, records horizontally):

Name	Address	City	State	Zip	phone
Bob Smith	1111 3rd St.	Denver	CO	11276	323-998-9987
June Day	220 8th Ave	Moline	IL	61265	309-762-1100
Tom Leaky	11 3rd St.	Milan	IL	61201	309-753-0098

etc.

It starts out just that simple. A Visual Basic program to maintain just this sort of minimal database would consist of nothing more than a form consisting of Labels and Text boxes designed to aid the user in data entry and a few buttons to facilitate database functions. The DEMO1 example program is a close approximation of this sort of application.

Database Field Types

Each field in a database has a corresponding data type. The available field types in the DataTable (version 1.1) 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.

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 data structures (Visual Basic User Defined Types). DataTable programmers receive field values from data table files as String values and write database field values to the DataTable control 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.

Indexes and Searching

Database files generally have some sort of indexing scheme in order to facilitate quick searching capabilities. As stated previously, the DataTable Custom Control (version 1.1) works with Paradox database files. Paradox database files have the capability of supporting multiple indexes. These database indexes are classified into two categories:

Primary indexes

Secondary indexes (maintained and non-maintained)

The Primary index is the default index used in database searches, however, you are able to create and use secondary indexes in your applications. In these database indexes, you specify **key** fields for the index. The **key** fields are the fields you wish to search on or order data by (Primary indexes must have all **key** fields one right after the other with the first key field being the first field in the database) . Primary indexes can have multiple **key** fields.

Just like searching for specific topics in a book, searching a database for specific information is done much more quickly when there is an index present. Indexes order data viewed in a database table. For example, if you have a database table with a single **key** field of type **Number (N)** in the Primary index. Database records viewed through that index will be ordered sequentially in ascending order based on the numeric values present in the **key** field i.e., 0,1,2,3,4,5, etc.

Database table files can be created through code when using the full featured DataTable product. The Working Model release of DataTable however does not support database table file creation through the use of code. A utility program called VBENGINE DATABASE TABLE MAKER is included in the Working Model so that you can create your own database tables. This utility is an interactive utility which is very easy to use. Creating your tables with the VBENGINE DATABASE TABLE MAKER utility is much easier than doing so through code.

Database indexing and searching are some of the more complicated concepts when first learning about database systems. DataTable programmer's who may need more details concerning indexes, searching via indexes, general searching techniques, and querying database tables should obtain a copy of the DataTable User's Manual.

Multi-User Environments

The DataTable Custom Control can be used in Local Area Network environments consisting of multiple users sharing the same database. The DataTable working model comes with file and record

locking facilities. DataTable **Action property** values specific to network file sharing environments are **LockRecord, UnlockRecord, LockFile, UnlockFile, GetUserName**, etc. Note: only record locking is supported in the Working Model. The GetUserName is also unavailable in the Working Model.

For a complete description of the concepts introduced in this section and information on other DataTable database related information please see the DataTable User's Manual. For specific information on the Paradox database file structure and concepts relevant specifically to the Paradox Engine see the Paradox Engine User's Guide available from Borland International.

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.

Distribution Note

When you create and distribute applications which use the DataTable custom control you should install the files **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

Left

*SearchMode

*FieldName	Name	*TableName
*FieldType	*NRecords	Tag
*FieldValue	*Reaction	Top
*IndexID	*RecordNumber	Visible
*KeySearch	*SaveEveryChange	

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 twenty-five DataTable Actions available for use in this Working Model release of the DataTable.

Remarks

The Action property settings are:

Setting	Description

0	None (No action)
1	AppendRecord
2	ClearRecord
3	CloseTable
4	DeleteRecord
5	FirstRecord
6	GetField
7	GetFieldType
8	GetRecord
9	GetRecordNumber
10	GotoRecord
11	InsertRecord
12	LastRecord
13	LockRecord
14	NRecords
15	NextRecord
16	OpenTable
17	PreviousRecord
18	PutBlank
19	PutField
20	RefreshTable
21	SearchField
22	SearchKey
23	UnlockRecord
24	UpdateRecord

DataType

Integer (Enumerated)

DataTable Actions

0-None

Description

This Action does nothing.

1-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.

2-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**

3-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

4-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.

5-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.

6-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** 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.

7-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

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.

8-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.

9-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.

10-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.

11-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.

12-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

13-LockRecord

Description

Locks the current database record.

Remarks

This Action locks the current record. The database table and it's 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

14-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.

15-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.

16-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, and CloseEngine.

17-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.

18-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.

19-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.

20-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.

21-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

22-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 and Database Searching in the **Database Fundamentals** section of this manual.

23-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 conditionss:

- 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 successfull **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

24-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 **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

AppendRecord, InsertRecord, DeleteRecord.

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

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 1.1) 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.

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

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

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

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

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

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)

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.

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

DataTable Example Programs

In this section detailed examples of how to use the DataTable in the Visual Basic programming language will be presented. Details concerning how to use the DataTable custom control to read and write data between Visual Basic programs and database files are covered in detail. Several example programs have been sent along as part of the working model distribution file set. These example programs are discussed here in detail.

This section is a subset of the same section in the registration copy of the DataTable User's Manual. There is not as much information concerning complex searching, relational models, and querying present here. There is however, sufficient information to get one started in DataTable programming. Specific examples of how to search with an index and how to search on a field are presented here. Also an example is given on how to fill in Visual Basic List and Combo boxes with data from a database table using the DataTable. For extensive information concerning database indexing and searching please see the registration copy of the DataTable User's Manual.

Installation Note: The example programs presented in this section (including the Visual Basic source code programs included with the working model) expect the files **CUSTOMER.DB** and **CUSTOMER.PX** files to be in the C:\ directory. If you wish to change this location do so in the example programs source code.

DEMO1

This is the first example program. It is very simple and illustrates a few basic DataTable programming concepts. It can be found in the working model distribution file set. The files DEMO1.MAK, DATATBL.BAS, DEMO1.FRM, and DATATBL.VBX constitute the file set for the DEMO1 example program. The source code is commented. (The database table and index used in the DEMO1 example program were created using the VBENGINE DATABASE TABLE MAKER utility program which is also included in the working model file distribution).

The DEMO1 example program is a very simple illustration of how easily a database application can be generated in Visual Basic using the DataTable control. The DEMO1 example program consists of one database table "C:\CUSTOMER.DB" with an index file "C:\CUSTOMER.PX". The structure of the C:\CUSTOMER.DB database is shown below.

Field	Type
Name	A50
Address	A50
City	A30
State	A2
Zip	A10
Phone	A14

The DEMO1.FRM form was designed to be a window into the database. Using this form, users

can view the customer data in the database on a record-by-record basis. There is a field on the form for every field in the database. Six Text controls are used to hold database field values, and six Label controls are used to label those fields for the users benefit. There are seven push button controls on the form for database manipulation:

- **Top** moves to the first record in the database.
- **Bottom** moves to the last record in the database
- **Previous** moves to the previous record in the database
- **Next** moves to the next record in the database
- **Update** updates the current record in the database
- **Insert** inserts the form data as a record in the database
- **Delete** deletes the current record from the database

There are two utility push button controls on the form:

- **Clear** clears all information from the form (blank form)
- **Quit** terminates the demo program

We will start the description of this example application in the DEMO1 DataTable control which is called **Customer**. Look at the property settings of this control. You will see that the name of the control is **Customer**. The database table for this control is **C:\CUSTOMER**. The only two other properties important at this stage of the game are **IndexID** and **SaveEveryChange**. (The IndexID property set to 0 indicates that all indexes will be opened with the PRIMARY index being used. The SaveEveryChange property is set to False meaning that data is buffered to disk. For more information on these properties see the DataTable Custom Control reference section of this document.)

We next see the following code in the Forms load procedure:

```
Sub Form_Load ()
```

```
'This code opens the database table (C:\CUSTOMER) and fills in the Visual Basic  
'Form with the information present in the first record in the database.
```

```
Customer.Action = OpenTable 'Open up the database table (Customer.TableName)  
FillForm 'Fill the form
```

```
End Sub
```

This is only two lines of code! In this two lines of code we have:

- Opened the database file
- Called a Visual Basic subroutine **FillForm** which will read the data from a record and place that data onto our form.

Now let's examine the **FillForm** subroutine to see what it takes to actually read data from the database and place it in our Visual Basic form. The **FillForm** subroutine is a subroutine present in the Form's general section. Here it is in it's entirety:

Sub FillForm ()

**'We need to get the current record in the database:
Customer.Action = GetRecord**

**'Ok, now lets get the values in the database fields for the
'record we just read in:**

**'We need to make a one-to-one relationship between the fields in the database table
'and the fields on our Visual Basic Form. We will read in the field values from the
'database record and place those values into the .TEXT property of our TEXT controls
'on our Visual Basic form. For each field of interest this is a three-step process:**

**'1. Specify the name of the database field of interest ---> DataTable.FieldName = "?????"
'2. Get the field value ---> DataTable.Action = GetField
'3. Place the field value in the TEXT control ---> Text1.TEXT = DataTable.FieldValue**

**'Thats all there is to it!
,**

**'You should make a FillForm like subroutine position independent. Notice that the first line of
'code in this subroutine reads in the current record. Use other routines to move around
'in the table (like the TOP,BOTTOM, NEXT and PREVIOUS buttons in this demo). Making the
'routine position independent means that it can be used and re-used at any time to fill in
'your form with information from any record in your database.**

'We will fill in the form now:

**Customer.FieldName = "Name"
Customer.Action = GetField
CustomerName.Text = Customer.FieldValue**

**Customer.FieldName = "Address"
Customer.Action = GetField
Address.Text = Customer.FieldValue**

**Customer.FieldName = "City"
Customer.Action = GetField
City.Text = Customer.FieldValue**

**Customer.FieldName = "State"
Customer.Action = GetField
State.Text = Customer.FieldValue**

**Customer.FieldName = "Zip"
Customer.Action = GetField
Zip.Text = Customer.FieldValue**

**Customer.FieldName = "Phone"
Customer.Action = GetField
Phone.Text = Customer.FieldValue**

End Sub

Notice that the **FillForm** subroutine is a general purpose subroutine. It simply reads in a record's worth of data and displays that data on our form. It does not in any way position the current record in the database. It reads in the current record and displays the field data on the form. The point here is that we will use other routines to move around in the database and once we position to the desired record we will call **FillForm** to display the information.

At this point in time, our DEMO1 program has opened up our Customer database, read in the first record and displayed the customer data on our form. The program is now waiting for the user to do something. Let's look at the top row of push button controls on our form:

Top	Bottom	Previous	Next
------------	---------------	-----------------	-------------

These push button controls are for movement in the database table. They let our DEMO1 user navigate through our database. Let's take a look at the code attached to each of these push button controls:

Sub TopButton_Click ()

'The following code moves to the first record in the data base
'and fills in our form with the information found in that first
'record

```
Customer.Action = FirstRecord
FillForm
End Sub
```

Sub BottomButton_Click ()

'The following code moves to the last record in the data base
'and fills in our form with the information found in that last
'record

```
Customer.Action = LastRecord
FillForm
End Sub
```

Sub PreviousButton_Click ()

'The following code moves to the previous record in the database table. If an
'error is encountered during the move to the record an error message is displayed.
'The DisplayError subroutine is an example of the structure of a generalized
'DataTable error handling routine.

'Once the repositioning to the previous record is successfull, we fill in our form
'with information from that record.

```
Customer.Action = PreviousRecord 'Moving to previous record
```

```
If (Customer.Reaction <> 0) Then      'If an error
    DisplayError (Customer.Reaction) 'Display the DataTable Error
    Exit Sub
Else                                  'If no error
```

```

FillForm      'Fill in our form with info from the record
End If

End Sub

```

```

Sub NextButton_Click ()

```

```

'The following code moves to the next record in the database table. If an
'error is encountered during the move to the next record an error message is displayed.
'The DisplayError subroutine is an example of the structure of a generalized
'DataTable error handling routine.
'

```

```

'Once the repositioning to the next record is successfull, we fill in our form
'with information from that record.

```

```

Customer.Action = NextRecord      'Moving to the next record in the database

If (Customer.Reaction <> 0) Then    'If an error
    DisplayError (Customer.Reaction) 'Display the DataTable error
    Exit Sub
Else                                'If no error
    FillForm                        'Fill in our form with the information in the record.
End If

End Sub

```

Pretty simple code! In essence, each one of these positional controls simply performs a single DataTable Action to reposition the database's current record pointer. Then calls the **FillForm** subroutine to read the data in and display it on our form. (The Next and Previous buttons also demonstrate how to incorporate a DataTable error handling routine.)

Now let's take a look at the **Delete** push button's code:

```

Sub DeleteButton_Click ()
'This code deletes the current record from the database
'The current record after the delete is performed, is the record immediately
'following the deleted record. We then fill in our form with info from that
'record.

Customer.Action = DeleteRecord    'We will delete the current record.
FillForm                      'Now fill in form with the current record.

End Sub

```

It doesn't take a lot of code to delete a record from the database. When this push button is clicked by the user, the record is deleted from the database by performing the DeleteRecord Action. When the record is deleted from the database the DataTable control automatically moves the database record pointer to the next available record so all we have to do is to call our **FillForm** subroutine to display the current database record.

Now we only have two more database related push button controls to look at **Update** and **Insert**.

```
Sub UpdateButton_Click ()  
UpdateRec  
End Sub
```

```
Sub InsertButton_Click ()  
InsertRec  
End Sub
```

That's it for the buttons themselves, now let's look at the two subroutines **UpdateRec** and **InsertRec** each present in the Form's general section:

```
Sub UpdateRec ()
```

```
'Here we transfer the information from the Visual Basic Form and update the  
'current record in the database table.
```

```
'To do this requires a three-step process for each field in the database table:
```

- '1. Specify the name of the field of interest ---> DataTable.FieldName
- '2. Specify the field value ---> DataTable.FieldValue = A String value
- '3. Put the field into the DataTable record ---> DataTable.Action = PutField

```
'We do this for each field in the database table.
```

```
'When we are finished we UPDATE the record into the  
'database table ---> DataTable.Action = UpdateRecord
```

```
Customer.FieldName = "Name"  
Customer.FieldValue = CustomerName.Text  
Customer.Action = PutField
```

```
Customer.FieldName = "Address"  
Customer.FieldValue = Address.Text  
Customer.Action = PutField
```

```
Customer.FieldName = "City"  
Customer.FieldValue = City.Text  
Customer.Action = PutField
```

```
Customer.FieldName = "State"  
Customer.FieldValue = State.Text  
Customer.Action = PutField
```

```
Customer.FieldName = "Zip"  
Customer.FieldValue = Zip.Text  
Customer.Action = PutField
```

```
Customer.FieldName = "Phone"  
Customer.FieldValue = Phone.Text  
Customer.Action = PutField
```

```
Customer.Action = UpdateRecord
```

```
End Sub
```

Sub InsertRec ()

**'Here we transfer the information from the Visual Basic Form and insert it
'into the database table.**

'To do this requires a three-step process for each field in the database table:

'1. Specify the name of the field of interest ---> DataTable.FieldName

'2. Specify the field value ---> DataTable.FieldValue = A String value

'3. Put the field into the DataTable record ---> DataTable.Action = PutField

'We do this for each field in the database table.

**'When we are finished we INSERT the record into the
'database table ---> DataTable.Action = InsertRecord**

Customer.FieldName = "Name"

Customer.FieldValue = CustomerName.Text

Customer.Action = PutField

Customer.FieldName = "Address"

Customer.FieldValue = Address.Text

Customer.Action = PutField

Customer.FieldName = "City"

Customer.FieldValue = City.Text

Customer.Action = PutField

Customer.FieldName = "State"

Customer.FieldValue = State.Text

Customer.Action = PutField

Customer.FieldName = "Zip"

Customer.FieldValue = Zip.Text

Customer.Action = PutField

Customer.FieldName = "Phone"

Customer.FieldValue = Phone.Text

Customer.Action = PutField

Customer.Action = InsertRecord

End Sub

Not too difficult is it! Well that's about it, a few more minor details to cover, like the **Clear** push button. All it does is clear the text values in the Form's Text controls. No DataTable Actions are associated with the **Clear** button. However, an important concept is associated with the **Quit** button. Remember way back at the beginning, when we set up the **DataTable** control's SaveEveryChange

property to buffer database changes too disk? Well before we quit the DEMO1 example program we want to make sure that any changes are indeed saved to the database disk file. This can be done at any time manually by performing the DataTable's **FlushBuffers** Action (not available in the Working Model release). But in DEMO1, we simply rely on the **CloseTable** Action to save all changes before closing the table.

Database Searching Techniques

The DEMO2 Example Program (Searching on a Specific Field)

The DEMO2 program is a modified version of the DEMO1 example program. DEMO2 is DEMO1 with one extra subroutine **CustomerName LostFocus()**. The CustomerName field is a window into our Customer database's Name field. The DEMO2 program is structured to accept keyboard input from the user and when the user types in a customer's name and leaves the CustomerName control (LostFocus), the Customer database is searched (on the NAME field) for the name typed in by the user. If the user typed name is not found, the remaining fields are cleared and we expect to receive information for a new customer. If the user typed name is found in the database, the remaining fields on the form are filled in with that customer's information.

Let's look at the code in the **CustomerName LostFocus** subroutine:

Sub CustomerName_LostFocus ()

'This code demonstrates how to search a database on a specific field for a given value.
'When you leave this field, this lost focus subroutine is executed.

'The value in the CustomerName.Text is then searched for in the database table.
'The database field searched is "Name".

'If the search is successfull the record is displayed on the form. If the search
'fails an error message is displayed, the form is cleared (with the typed in value)
'replaced in the CustomerName.Text expecting to get a new customer's information.

Dim TheNameTyped As String 'Temporary variable to hold Customer's name.

Customer.SearchMode = SEARCHFIRST

Customer.FieldName = "Name"
Customer.FieldValue = CustomerName.Text
Customer.Action = PutField

Customer.Action = SearchField
If (Customer.Reaction <> 0) Then
 DisplayError (Customer.Reaction)
 TheNameTyped = CustomerName.Text
 ClearButton_Click
 CustomerName.Text = TheNameTyped
Exit Sub

```

Else
    FillForm
End If

End Sub

```

The DEMO3 Example Program (Searching with an Index)

The DEMO2 example program shows how you can search a database on a specific field for a given value. The DEMO3 program is a modified version of the DEMO1 example program. DEMO3 is DEMO1 with one extra subroutine **CustomerName LostFocus()**. The CustomerName field is a window into our Customer database's Name field. The DEMO3 program is structured to accept keyboard input from the user and when the user types in a customer's name and leaves the CustomerName control (LostFocus), the Customer database is searched (on the PRIMARY index) for the name typed in by the user. If the user typed name is not found, the remaining fields are cleared and we expect to receive information for a new customer. If the user typed name is found in the database, the remaining fields on the form are filled in with that customer's information.

Let's look at the code in the **CustomerName LostFocus** subroutine:

```

Sub CustomerName_LostFocus ()

```

```

    'This code demonstrates how to search a database on a specific key.
    'When you leave this field, this lost focus subroutine is executed.

```

```

    'The value in the CustomerName.Text is then searched for in the database table.
    'The database field searched is "Name" (The only key field in this example).

```

```

    'If the search is successful the record is displayed on the form. If the search
    'fails an error message is displayed, the form is cleared (with the typed in value)
    'replaced in the CustomerName.Text expecting to get a new customer's information.

```

```

    Dim TheNameTyped As String    'Temporary variable to hold Customer's name.

```

```

    Customer.SearchMode = SEARCHFIRST
    Customer.KeySearch = 1        'The first and only field in PRIMARY index

```

```

    Customer.FieldName = "Name"    'The first and only field of the key
    Customer.FieldValue = CustomerName.Text 'The value to search for
    Customer.Action = PutField      'Submit the search criteria

```

```

    Customer.Action = SearchKey    'Perform the search

```

```

    If (Customer.Reaction <> 0) Then    'If an error
        DisplayError (Customer.Reaction) 'Display the error
        TheNameTyped = CustomerName.Text 'Assume new customer, temp store for new
customer name
        ClearButton_Click              'Clear the form
        CustomerName.Text = TheNameTyped 'Place the new customer name back in the form

```

```

Exit Sub          'Now let the user type in the new info
Else              'Search was successfull
  FillForm        'Fill in the form with customer info
End If

End Sub

```

Note: Searching techniques using the DataTable are discussed in greater detail in the DataTable User's Manual.

The last example program in the DataTable (version 1.1) working model distribution file set is the DEMO4 example program. The DEMO4 program is pretty much the same as the DEMO3 program but it includes an example on how to read data from a database table and place that data into a combo box for database sourced pick lists. The DEMO4 program will not be examined in detail here, the source code is well commented. Look for the new subroutines **FillCustomerCombo** and **EmptyCustomerCombo** in the Form's general section. For more DataTable example programs and more detailed information on DataTable programming place your order for a registered copy today. From time-to-time new versions of the DataTable working model will be released. New example programs covering different aspects of DataTable programming will be distributed therein.

DataTable / PARADOX ENGINE ERROR CODES

Error Code	Description
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
70	No more file handles available
72	No more table handles available
73	Invalid date given
40	

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