

CDB For Windows

Copyright (C) 1991 by Daytris
All rights reserved

Revised: May 16th, 1991

Table Of Contents

Introduction

Overview		
	4	
Obtaining CDB For Windows		5
Future Enhancements		5
Contacting the Developers		
	6	

Getting Started

Unpacking		
	7	
Sample Application		9

The Network Database Model

Introduction		10
The Relational Model		10
What is the Network Model?		10

The Data Definition Language

Introduction		11
Using sampledb.ddl as an Example		11
prefix		
	12	
connect		12
key		
	16	
DDL Limitations		18

Database Currency

What is Currency?		19
An Example		19
keydba		
	20	
datadba		21
Differences Between Find and Get Function Calls		21
Storing Currency Tables		21
Deleting a current record		22
Updating a current record		23

Database Internals

Client-Server Implementation		24
dbserver.exe		24
dbtalk?.lib		24
Memory Requirements		25
database server		25
dbtalk library		26
Database Files		26
File Naming		26

Key File Layout	27
Data File Layout	28
Modifying Internal Definitions	29

Utilities

DDL.P.EXE	31	
DBDLIST.EXE		31

Using the C-API

Introduction		32
Functions by Category		32
API Function Descriptions (in alphabetical order)		35

Error Messages

CDB Run-Time Error Messages		120
DDL.P Error Messages		122

Introduction

Overview

CDB For Windows is a sophisticated database toolkit for Microsoft Windows programmers. CDB For Windows includes the following features:

- Quick data access through a sophisticated multi-key ISAM implementation.
- Multiple data models. Both relational and network data models are implemented in CDB For Windows. The network model gives the developer the ability to create relationships between records without storing unique keys in those records.
- Data Definition Language (DDL) for defining database layouts. The DDL is compiled into a binary format which is used by the database server as a roadmap. Using this concept, a developer can define and re-define a database with minimal effort and absolutely no code changes. The DDL is patterned after C for ease of programming.
- Multiple database access. More than one database can be accessed concurrently by an application.
- Client-server implementation. When a database is opened, an instance of the database server executable is loaded and is responsible for handling all client requests.
- Over 40 predefined database function calls for complete control of the database.
- Portability. CDB For Windows is written entirely in C for portability and source code is available. Versions of CDB are also available for MS-DOS and UNIX platforms. Contact the developers for more information about these products.
- C++ compatibility. The library is callable from both C and C++.
- Automatic re-use of deleted database space. There is no need for database reorganization. Deleted space is automatically reused by CDB For Windows.
- Low overhead. The CDB For Windows database engine requires an extremely small amount of memory to operate. In the SAMPLE application included, the database server, when loaded, takes up only 57K of memory.
- Royalty-free distribution rights. Whether you have one customer or thousands, you pay for CDB just once for each environment that you are using.

CDB For Windows follows all of the Microsoft guidelines for appropriate use of memory under MS-Windows. All memory segment locking and unlocking is on an "as needed" basis, transparent to the user, and in full compliance with Windows requirements.

The client-server implementation in CDB For Windows enables multiple applications to utilize the same database code segments thereby reducing the memory consumption under MS-Windows.

Obtaining CDB For Windows

Refer to the document included in this release, ORDER.TXT, to order CDB For Windows. Your purchase will include CDB For Windows libraries, utilities, royalty-free use of library functions, full library and utility source code and make files. A printed manual will also be included.

Future Enhancements

Listed below are some of the enhancements planned for CDB For Windows. Any suggestions would also be greatly appreciated.

- The DBTALK library will become a Dynamic Link Library.
- Multi-application database access. More than one application under MS-Windows will have the ability to share the same database.
- Multi-user database access. Networks and/or protocols to be supported are currently undefined.
- Structure Query Language (SQL) interface.
- Performance enhancements.
- OS/2 version.

Contacting the Developers

CDB for Windows was developed solely by Daytris. If you have a question about the product or any suggestions please contact us at the phone number listed below. Or if you prefer, you can send us electronic mail on any of the information services listed.

Daytris
81 Bright Street, Suite 1E
Jersey City, NJ 07302
201-200-0018

CompuServe:	72500,1426
BIX:	daytris
GEnie:	t.fearn

Getting Started

Unpacking

CDB For Windows is distributed in self-extracting ZIP files. When you first unpack your software, you may want to verify that you have a complete set. If you have the Test Drive version, you should have the files included in CDBW.EXE. If you have purchased CDB For Windows, you should have the files included in both CDBW.EXE and CDBWSRC.EXE. The contents of each are listed below.

CDBW.EXE

<u>File Name</u>	<u>Description</u>
CDBW.WRI	This document.
ORDER.TXT	Order form for CDB For Windows.
REGISTER.TXT	Registration form for CDB For Windows.
DBTALKS.LIB	Small model DBTALK library.
DBTALKM.LIB	Medium model DBTALK library.
DBSERVER.EXE	Database server executable.
DBDLIST.EXE	Database Definition (DBD) file display utility.
DDL.P.EXE	Database Definition Language Parser.
SAMPLE.EXE	Sample CDB For Windows application.
SAMPLE.C	Module for SAMPLE.EXE. Contains WinMain and setup functions.
SABOUT.C	Module for SAMPLE.EXE. Contains functions to control the About dialog.
SCLIENT.C	Module for SAMPLE.EXE. Contains functions to control the Client dialog.
SADDRESS.C	Module for SAMPLE.EXE. Contains functions to control the Address dialog.
SLISTBOX.C	Module for SAMPLE.EXE. Contains functions that maintain the client and address listbox controls.
SERROR.C	Module for SAMPLE.EXE. Contains error handling functionality.
SAMPLE.MAK	SAMPLE make file. Microsoft NMAKE and UNIX make compatible.
SAMPLE	SAMPLE make file. Microsoft MAKE compatible.
BSAMPLE.MAK	SAMPLE make file. Borland MAKE compatible.
BSAMPLE.LNK	SAMPLE link response file. Borland MAKE compatible.
SAMPLE.DEF	SAMPLE module-definition file.
SAMPLE.RC	SAMPLE resource-script file.
SAMPLE.MNU	SAMPLE menu definition file.
SAMPLE.DLG	SAMPLE dialog definition file.
SAMPLE.H	SAMPLE header file.
SAMPLEDB.DDL	SAMPLE data definition language file.
SAMPLEDB.DBD	SAMPLE database definition file. Binary.
SAMPLEDB.H	SAMPLE database header file.
DBMGR.H	CDB For Windows header file.

CDBWSRC.EXE

<u>File Name</u>	<u>Description</u>
DBSERVER.MAK	Make file for the database server program, DBSERVER.EXE. Microsoft NMAKE and UNIX make compatible.
DBSERVER.DEF	DBSERVER module-definition file.
DBSERVER.RC	DBSERVER resource-script file.
DBSERVER.C	DBSERVER source file.
DBMGR.C	.
DBADD.C	.
DBUPD.C	.
DBDEL.C	.
DBFIND.C	.
DBGET.C	.
DBCURR.C	.
DBFILE.C	.
DBPAGE.C	.
DBSLOT.C	.
DBFUNCS.C	.
DBLFUNCS.C	.
DBMGR.H	CDB For Windows header file. Used by DBSERVER, DBTALK, DDLP, DBDLIST, and client applications.
DBXTRN.H	DBSERVER external definitions header file.
STDINC.H	Header file that includes other header files used in CDB For Windows.
CALLTYPE.H	Header file categorizing CDB For Windows calls. Used by DBSERVER and DBTALK.
DBTALK.MAK	Make file for the DBTALK library. Microsoft NMAKE and UNIX make compatible.
DBTALK.C	DBTALK library source file.
DBCALL.C	.
DDL.MAK	Make file for DDLP. Microsoft NMAKE and UNIX make compatible.
MAIN.C	DDL source file.
DDL.C	.
PARSE.C	.
ERROR.C	.
DDL.H	DDL header file.
DBDLIST.MAK	Make file for DBDLIST. Microsoft NMAKE and UNIX make compatible.
DBDLIST.C	DBDLIST source file.

Sample Application

The sample application, SAMPLE.EXE, included in this release is written in C and conforms to the user-interface style recommended by Microsoft for Windows applications.

SAMPLE.EXE is a simple client list program. It is important not for what it can do, but for what it provides: a template for developing applications that use the CDB For Windows API.

SAMPLE will help you better understand how to use the API. Full source, header, module-definition, resource, and make files are provided.

The Network Database Model

Introduction

CDB For Windows provides both relational and network model features. The use of both models in a data design can greatly increase the performance of your database. For those of you who are already familiar with the relational database concepts, you will find the network model implementation very refreshing. For those of you who aren't familiar with the relational model, a very brief description of relational concepts follows.

The Relational Model

In a relational database, data is stored in a series of tables. Each table consists of a number of columns, which identify a particular type of data, and rows, which correspond to a particular record in the table.

Individual records can be retrieved using the key fields defined for the table. If the developer has the desire to make an association between two tables, unique key fields must be defined in both records and unique data must be stored for retrieval to take place.

What is the Network Model?

This model allows you to define relationships between records through constructs called sets. A set defines a one-to-many relationship between two tables.

In a relational model, records can only be related (connected) by storing unique keys in both tables. This method creates additional unwanted overhead. Duplicate data is stored in both records and duplicate indexes must be managed.

Using the network model, records are connected by directly storing data pointers inside the record. Where the relational model requires multiple disk accesses to locate a related record, the network model allows the record to be located in a single disk access. Disk space is also saved when sets are used because no index is required.

Another advantage of using the network model is the flexibility of owner/member relationships. A record may own multiple record types. A record may also be owned by multiple owner records. An example of this would be a 'client' record owning 'invoice' records and also owning 'address' records. In turn, an 'invoice' could also own 'address' records, perhaps a shipping and billing address. This kind of flexibility gives the developer the power to define complex data relationships with relative ease.

The Data Definition Language

Introduction

The Data Definition Language is use for defining a database model. The language is basically a superset of C structure definitions. If you are familiar with defining C structures, the DDL should be very easy to pick up on.

The DbOpen function call loads a binary image of a DDL file. The binary image is created by compiling the DDL file into a DBD (Database Definition) format. A DDL compiler is included with this release, DDL.P.EXE (Data Definition Language Parser).

Using SAMPLEDB.DDL as an Example

The SAMPLE program included with release contains a DDL, SAMPLEDB.DDL. We will use this DDL as an example.

```
/* sampledb.ddl */
```

```
prefix ABC;
```

```
struct client
{
    connect          address key szStreet;
    key long         IClientNbr;
    key char         szName[31];
    char            szDescription[61];
    double          dBalance;
};
```

```
struct address
{
    char            szStreet[31];
    char            szCity[21];
    char            szState[3];
    key char        szZip[11];
    key char        szTelephone[13];
    char            szFax[13];
};
```

```
struct setup
{
    key long        INextClientNbr;
};
```

Notice the close resemblance to C structure definitions. The only differences are the **prefix**, **connect**, and **key** words.

prefix

The prefix is used internally by the database server (DBSERVER.EXE) when a new database file must be created. In the SAMPLEDB.DDL shown above, the prefix is "ABC". By defining the prefix as "ABC", we are telling the database server to use "ABC" as the first 3 characters of any file that is created for the SAMPLEDB database.

The prefix can be from 1 to 4 characters in length. If a prefix is not defined, a default prefix, "TEST", is used.

For more information about the CDB For Windows database file naming conventions, refer to the 'Database File Names' section in this manual.

connect

When using the connect keyword, you are taking advantage of the network database model implementation of CDB For Windows. Network model concepts can greatly increase the performance and efficiency of your database.

The connect keyword defines a relationship between two records.

```
struct client
{
    connect          address key szStreet;
    .
    .
};
```

In this example, we are defining a relationship between the client record and the address record. The client record will be an owner of the address record. The address record is a member of the client record. For now, ignore the 'key szStreet' part of the connect phrase.

By declaring this set relationship, we now have the capability to make connections between client and address records using the DbSet... function calls. A client may own 0, 1 or many address records. Without the network model concepts that we have just shown you, to make connections between two records would require the storage of a unique key in each individual record.

```

void Function(HANDLE hDb)
{
    static CLIENT client = {1000L,"Daytris","A software company",0.00};
    static ADDRESS address = {"81 Bright Street, Suite 1E","Jersey
        City","NJ","07302","201-200-0018",""};

    XDbRecordAdd( hDb, "client", &client, sizeof(CLIENT));
    XDbRecordAdd( hDb, "address", &address, sizeof(ADDRESS));
    DbSetAdd( "client", "address");
}

```

The example above shows how to make a set connection between two records by using the DbSetAdd function. After the function call, the client "Daytris" is the owner of 1 address record. This address record can be retrieved using the DbSetGetFirst or XDbSetGetFirst calls:

```
XDbSetGetFirst( hDb, "client", "address", &address, sizeof(ADDRESS));
```

Now lets add another address record to the set:

```

void Function(HANDLE hDb)
{
    long lKey = 1000L;
    static ADDRESS address = {"30 Broad Street","New York","NY","10015","212-555-
        1212","212-555-1212"};

    /* Make client #1000 current */
    XDbRecordFindByKey( hDb, "client", &lKey, sizeof(LONG));

    /* Add another member */
    XDbRecordAdd( hDb, "address", &address, sizeof(ADDRESS));
    DbSetAdd( "client", "address");
}

```

The client "Daytris" now owns 2 address records. We can use the DbSetGetFirst, DbSetGetLast, DbSetGetNext, DbSetGetPrev, or any of the extended versions of these API calls to retrieve any of the address records in the set.

Now lets take a look at how the sets are ordered. If we make a DbSetGetFirst call after adding the sets shown above, which address record would be returned? Lets return to our original DDL example:

```

struct client
{
    connect                address key szStreet;
    .
    .
};

```

Member records can be ordered two ways. By the order in which they are added, or by a field in the member record. In our example, the set order is by the szStreet field in the address record. Therefore, a DbSetGetFirst(hDb, "client", "address", ...) call would return the "30 Broad Street" address record. If the connect address phrase were defined without a key:

```
struct client
{
    connect          address;
    .
    .
};
```

the address members would be stored in the order that they were added. Therefore, a DbSetGetFirst(hDb, "client", "address", ...) call would return the "81 Bright Street, Suite 1E" address because this address was added first.

A record can have more than one member. A record can also be owned by more than one owner. To illustrate this, lets take the SAMPLEDB.DDL and expand it to include invoicing capabilities.

```
/* sampledb.ddl - with invoicing */
```

```
prefix ABC;
```

```
struct client
{
    connect          address key szStreet;
    connect          invoice;
    key long         IClientNbr;
    key char         szName[31];
    char             szDescription[61];
    double           dBalance;
};
```

```
struct address
{
    char             szStreet[31];
    char             szCity[21];
    char             szState[3];
    key char         szZip[11];
    key char         szTelephone[13];
    char             szFax[13];
};
```

```

struct invoice
{
    connect          address;
    connect          invoiceline;
    key long         lInvoiceNbr;
    long             lDate;
    double           dTotalPrice;
};

struct invoiceline
{
    long             lQuantity;
    char             szDescription[31];
    double           dUnitPrice;
    double           dLinePrice;
};

struct setup
{
    key long         lNextClientNbr;
    long            lNextInvoiceNbr;
};

```

In this example, a client record can own multiple address records and multiple invoice records. This makes sense because a client could have more than one address, i.e. a shipping and billing address. The client could also have more than one invoice if more than one order is placed.

Also in this example, an owner/member relationship exists between the invoice and address records. If our invoice has both 'ship to' and 'bill to' addresses, the shipping address could be stored as the first member in the set and the billing address could be stored as the next member.

A variable number of line items could exist on an invoice. This is the reason for the invoiceline record and its relationship with the invoice. An invoice record will own its invoice lines.

The following example illustrates how all data pertaining to a specific invoice might be retrieved. Note: it is suggested that the CDB return values be taken more seriously than illustrated below.

```

typedef struct invoice INVOICE;
typedef struct invoiceline INVOICELINE
typedef struct client CLIENT;
typedef struct address ADDRESS;

```

```

void Function(HANDLE hDb)
{
    long lKey = 2000L;
    DWORD dwStatus;
    INVOICE invoice;
    INVOICELINE invoiceline;
    CLIENT client;
    ADDRESS shipaddress;
    ADDRESS billaddress;

    /* Get invoice #2000 */
    XDbRecordGetByKey( hDb, "invoice", "lInvoiceNbr", &lKey, &sizeof(LONG),
        &invoice, sizeof( INVOICE));

    /* Get the client that owns the invoice */
    XDbSetGetOwner( hDb, "client", "invoice", &client, sizeof( CLIENT));

    /* Get the 'ship to' and 'bill to' addresses */
    XDbSetGetFirst( hDb, "invoice", "address", &shipaddress, sizeof( ADDRESS));
    XDbSetGetNext( hDb, "invoice", "address", &billaddress, sizeof( ADDRESS));

    /* Retrieve all invoice lines (assuming at least 1 line) */
    dwStatus = XDbSetGetFirst( hDb, "invoice", "invoiceline", &invoiceline,
        sizeof( INVOICELINE));
    while( dwStatus != E_NONEXT)
    {
        /* Store the line */

        /* Get the next line */
        dwStatus = XDbSetGetNext( hDb, "invoice", "invoiceline", &invoiceline,
            sizeof( INVOICELINE));
    }
}

```

This example illustrates some of the capabilities that you have with set relationships. The possibilities are endless.

key

The key word is used for defining key fields in records and key fields to be used in set relationships. See the **connect** section directly preceding this section for more details about the key fields in set relationships.

Key fields are stored in ascending order in slots on pages in a key file. The key file is made up of a series of linked pages.

```

struct client
{
    connect          address key szStreet;
    key long         IClientNbr;
    char             szName[31];
    char             szDescription[61];
    double           dBalance;
};

```

This DDL structure definition contains only one key field, "IClientNbr". Therefore all pages in the corresponding key file will contain slots of sorted client numbers.

```

struct client
{
    connect          address key szStreet;
    key long         IClientNbr;
    key char         szName[31];
    char             szDescription[61];
    double           dBalance;
};

```

The DDL structure definition now contains two key fields, "IClientNbr" and "szName". Therefore two types of key pages will exist in the key file for this record type. Some pages will contain slots of sorted client numbers and other pages will contain slots of sorted client names. The data stored on the key file pages is directly related to the number of key fields defined in the DDL file.

To maximize the efficiency of your database, it is suggested that you use as few key fields as possible. The maximum number of key fields allowed in a record is defined as MAXKEY in DBMGR.H. It is currently set to 8. See 'Modifying the Database Internals' section for more information about MAXKEY.

If a structure is defined without a key field, the only way to access a record of this type is with a set relationship. The structure defined without a key field must be a member of another record.

```

struct client
{
    connect          address key szStreet;
    key long         IClientNbr;
    key char         szName[31];
    char             szDescription[61];
    double           dBalance;
};

```

```

struct address
{
    char          szStreet[31];
    char          szCity[21];
    char          szState[3];
    char          szZip[11];
    char          szTelephone[13];
    char          szFax[13];
};

```

In this example, the address record contains no key fields. Therefore, the address record cannot be accessed using any DbRecord... function calls because these functions require a key field as a parameter. However, the address is a member of the client record. Therefore, it could be accessed with the DbSetGet... function calls, provided a relationship exists.

DDL Limitations

The Data Definition Language does not currently support the definition of structures or unions defined from within a structure. Example:

```

struct client
{
    struct address addr;
    key long        IClientNbr;
    key char        szName[31];
    char           szDescription[61];
    double          dBalance;
};

```

These deficiencies will be supported in a later release of CDB For Windows. A way to get around this problem for now is to allocate enough space as a char field for the structure or union that could not be included. Example (assuming the address structure length is 92 bytes):

```

struct client
{
    char          addr[92];
    key long        IClientNbr;
    key char        szName[31];
    char           szDescription[61];
    double          dBalance;
};

```

After DDLP compilation, modify the C header file output by DDLP to include the proper structure definition.

Database Currency

What is Currency?

Currency refers to the record position in a database key file. It is very similar to the file pointer in an open file. For example, when you first open a file using the C run-time library "open" function, the file pointer points to the first byte in the file (it could point to the last byte depending on how its opened). After the file is open, you can seek to different positions in the file and read or write data. The file pointer position is kept internally by the operating system. You could think of this position as the current position or "currency".

In CDB For Windows, the concepts are very similar. Each record structure defined in a DDL will have an associated currency table when this database is opened.

An Example:

```
/* sampledb.ddl */

prefix ABC;

struct client
{
    connect          address key szStreet;
    key long         IClientNbr;
    key char         szName[31];
    char            szDescription[61];
    double          dBalance;
};

struct address
{
    char            szStreet[31];
    char            szCity[21];
    char            szState[3];
    key char        szZip[11];
    key char        szTelephone[13];
    char            szFax[13];
};

struct setup
{
    key long        INextClientNbr;
};
```

When this database is opened using DbOpen, 3 currency tables will be initialized to zero. One for each record type: 'client', 'address', and 'setup'. The currency table contains the following format:

```
struct currency_index
{
    struct
    {
        UINT          page;
        UINT          slot;
    } keydba[MAXKEY]; /* Array of key dba's */
    ULONG          datadba; /* Data database address */
};
```

keydba

The currency table consists of two parts; a key currency (keydba) and a data record currency (datadba). A keydba exists for each key defined in the record table. In the example defined above, the 'client' record would use the first two keydba structures in the currency_index table for key currency storage. Records that do not have any keys defined would not make use of the keydba part of the currency_index.

Lets say that we have three 'client' records in our database. The contents of each are as follows:

```
Record 1:          1000L,"Daytris","Software Development",0.00
Record 2:          1001L,"Microsoft","Software Development",10000.00
Record 3:          1002L,"CompuServe","Computer Services",100.00
```

When the database is opened, the currency_index for the 'client' record, as well as all other records, is null. In other words, "the client record does not have currency". If we were to issue a:

```
DbRecordFindNext( hDb, "client", "IClientNbr");
```

at this time, an E_NONEXT return value would result. There is no next record to find! However, if we were to issue a:

```
DbRecordFindFirst( hDb, "client", "IClientNbr");
```

the return value would be 0L indicating a successful call. After this call, the keydba structure within the currency_index for the 'client' record would contain the appropriate page and slot number of the first record for the "IClientNbr" index. In this case, the keydba[0] structure within the 'client' currency_index would point to client number 1000L, Daytris.

If we were to now issue a:

```
DbRecordFindNext( hDb, "client", "IClientNbr");
```

the keydba[0] structure within the 'client' currency_index would point to the next client sorted by "IClientNbr". In our example, it would point to 1001L, Microsoft.

Keep in mind that we are not retrieving any records, we are only setting currency for the 'client' record type. If we would want to retrieve a record, we would use the DbRecordGet... function calls. Using the DbRecordFind... function calls we can essentially "seek" to positions within the database based on any index field within a record type.

datadba

The datadba field in the currency_index is used for "set" currency. When we issue a DbSetFind.. function call, the datadba is used to locate the current set record. The datadba field contains the actual slot number of the current record. "Next" and "previous" set pointers are stored at the beginning of each data slot in a data file.

Differences Between Find and Get Function Calls

The DbRecordFind... and DbSetFind... function calls only set currency for a specific record type. They do not retrieve records. You may wish to think of the Find function calls as performing the same task as the C run-time lseek function. Essentially, we are seeking to a position in the database.

If you wish to retrieve a record, use the DbRecordGet... or the DbSetGet... function calls.

Note: The DbRecordGet... and DbSetGet... function calls call their Find counterparts first, and then retrieve the current record. For example, the DbRecordGetFirst function will perform a DbRecordFindFirst and then a DbRecordGetCurrent function call.

Storing Currency Tables

You can retrieve a copy of the current currency_index for each record defined in the DDL. Why would you want to do this?

Lets suppose that you have a database that contains hundreds of 'client' records. Your application must be able to display these 'client' records in a small window, but you don't have enough memory to keep all of the 'client' records resident. Or it may be a waste of memory to do so. This is where storing currency tables becomes necessary.

As previously explained, each record type defined in a DDL has an associated currency table. The contents of a currency table can be retrieved or updated at any time. Therefore, in the example explained above, we could retrieve a window of 'client' records along with their associated currency_index tables. Example:

```

DWORD GetWindowOfClients( HANDLE hDb, BOOL bFirstTime)
{
    register short i;
    DWORD dwStatus;
    struct currency_index currency;

    for( i=0 ; i<WINDOW_LINES ; i++)
    {
        /* Get the record */
        if( bFirstTime)
        {
            bFirstTime = FALSE;
            dwStatus = XDbRecordGetFirst( hDb, "client", "IClientNbr", &client,
                sizeof(CLIENT));
        }
        else
            dwStatus = XDbRecordGetNext( hDb, "client", "IClientNbr", &client,
                sizeof(CLIENT));
        if( dwStatus)
            return dwStatus;

        /* Get the currency table */
        dwStatus = XDbRecordGetCurrency( hDb, "client", &currency, sizeof( struct
            currency_index));

        /* Put the record in a window and store along with it the associated currency
            table */
    }
}

```

After calling this routine, we have a window of 'client' records. For each 'client' record we also have an associated currency table. If the user were to select a specific 'client' in the window, we could retrieve this 'client' with the following database calls:

```

XDbRecordUpdCurrency( hDb, "client", &currency, sizeof( struct currency_index));
XDbRecordGetCurrent( hDb, "client", "IClientNbr", &client, sizeof(CLIENT));

```

The "currency" structure passed in the XDbRecordUpdCurrency call represents the currency_index of the selected 'client' record. Because we have a currency table stored for each 'client' record in the window, we can retrieve any record in the window using this method.

Deleting a current record

Beware when deleting a current record. If you are storing a series of currency tables as we have done in the example explained above, deleting a current record will invalidate currency tables that followed this record. Suppose we have a window of 'client' records:

Record 1: 1010L,"ABC Corp.,""Diskette Manufacturer",0.00
Record 2: 1011L,"XYZ Corp.,""Hard Drive Manufacturer",0.00
Record 3: 1012L,"BYTE Magazine","Software Publication",0.00
Record 4: 1013L,"Sharp","Electronics",0.00
Record 5: 1014L,"Collins","Radio Electronics",0.00

We have also stored currency tables associated with each 'client' record in the window.

If for example we delete Record 2, "XYZ Corp.", the currency tables associated with Records 3, 4, and 5 will now be invalid. Remember that currency tables contain the page and slot of a data item. If a record is deleted, the key fields are removed from their associated pages. The data (slots) on a key page are compressed to be contiguous. Therefore, key fields stored after the deleted record will be moved up 1 slot. Or possibly, if the slot is the last slot on a page, the page will be removed entirely.

To avoid retaining invalid currency tables in memory after deleting a record, currency tables should be re-retrieved after the deletion. To do this, start the retrieval with the record before the deleted record. In the example above, Record 2 is being deleted. After the deletion, restore the Record 1 currency table (using DbRecordUpdCurrency) and re-retrieve next records and corresponding currency tables (using DbRecordGetCurrency) until the window is full.

Updating a current record

An update, like the delete described above, can create similar problems. This is only a problem if the key field that was used for the retrieval of records is updated. In this case, the slots could be rearranged in an order unknown to the calling application. The only way to solve this problem, is to re-retrieve the records and their associated currency tables from the beginning after the update takes place.

Database Internals

Client-Server Implementation

CDB For Windows is implemented using a client-server architecture. The client is your application. The server is dbserver.exe

dbserver.exe

This executable is the database server program. When a database is opened using DbOpen, dbserver.exe is spawned. For each DbOpen call, an instance of dbserver.exe is loaded. Using this method, separate data and stack segments are loaded for each server required, but the code segments are shared by all servers. This significantly reduces the memory requirements of the database server.

The database handle returned from the DbOpen call represents the handle to the top window of the dbserver.exe instance. All subsequent calls to this database are identified by the handle.

The dbserver.exe is spawned using a LoadModule call with the SW_HIDE parameter. The server is hidden for a number of reasons, the most important being to prevent the termination of a database server by a user.

dbtalk?.lib

The DBTALK library comes in two forms. DBTALKS.LIB, small model library, and DBTALKM.LIB, medium model library. The library contains the API functions necessary for database management. The library is currently static. In a future release, DBTALK will be implemented as a Dynamic Link Library (DLL).

The DBTALK library communicates with the server by sending a message via the SendMessage function call. Before a message is sent, a table is built that describes the client request. The table format is described below:

```
typedef struct sCDBTALK
{
    WORD                wCall;
    char                szName1[67];
    char                szName2[32];
    HANDLE              hData1;
    HANDLE              hData2;
} CDBTALK;
```

Field	Description
<i>wCall</i>	The type of client request: D_OPEN, D_CLOSE, D_ADD_RECORD, D_DELETE_RECORD, etc.
<i>szName1</i>	A string location. Could be a record name, field name, database name, etc.
<i>szName2</i>	A string location. Could be a record name, field name, database name, etc.
<i>hData1</i>	Could contain a handle to a record to be added, a key value, a storage area, etc.
<i>hData2</i>	Could contain a handle to a record to be added, a key value, a storage area, etc.

The table space is allocated from global memory, filled and then sent to the database server. The database server responds to the request, and frees the table before returning control back to the client.

A database is closed with the DbClose function call. After the database server processes this call, the DBTALK library posts a WM_DESTROY to the server, terminating the instance.

Memory Requirements

CDB For Windows requires a very small amount of memory to operate. Memory is allocated by the the database server and the dbtalk library.

database server

As described above in the 'client-server implementation' section, after a DbOpen call, an instance of dbserver.exe is loaded. The database server memory requirements are as follows:

code segment:	31K	
data, local heap, stack:	26K	

Total memory:	57K	

The code segment is loaded only for the first instance of database server. Windows manages the sharing of the code segment for multiple instances of the database server. Therefore, the memory requirements are 57K for the first database opened, and 26K for databases opened thereafter.

The database server allocates all memory required for data access and manipulation during the DbOpen call. Memory is NOT allocated by the database server during the execution of any other database call.

A variable amount of memory is allocated beyond what has been described. Space for the DBD record, owner, member, and field tables. To calculate the amount of memory that will be required by the DBD, use the following formula:

$$\text{DBD memory consumption} = (\text{number of records} * 46) + (\text{number of owners} * 4) + (\text{number of members} * 8) + (\text{number of fields} * 40)$$

The DBDLIST utility included in this release will display the number of records, owners, members, and fields in a DBD. Example: the SAMPLEDB.DBD included in this release will consume 590 bytes when opened.

dbtalk library

The dbtalk library resides as a static library. In a future release, the dbtalk library will become dynamic. The library consists mostly of code and consumes approximately 5K of space.

Database Files

Database records are organized in data and key files. Each record type defined in the DDL will have an associated data file. If any key fields exist in this record, the record type will also have a key file. Key files have a .key extension while data files have a .dat extension.

File Naming

Lets take a look at the SAMPLEDB.DDL included with this release:

```
/* sampledb.ddl */
```

```
prefix ABC;
```

```
struct client
```

```
{
    connect          address key szStreet;
    key long         IClientNbr;
    key char         szName[31];
    char            szDescription[61];
    double          dBalance;
};
```

```
struct address
```

```
{
    char            szStreet[31];
    char            szCity[21];
    char            szState[3];
    key char        szZip[11];
    key char        szTelephone[13];
    char            szFax[13];
};
```

```

struct setup
{
    key long          INextClientNbr;
};

```

Key and data files are not created until the first record of a specific record type is added. For example, when the first 'client' record is added to the database, a data and key file will be created. The names used for the .DAT and .KEY file are derived as follows:

```

sprintf( szKeyFile, "%s%4.4d", szPrefix, nOrderInDBD);
sprintf( szDatFile, "%s%4.4d", szPrefix, nOrderInDBD);

```

The `szPrefix` is the prefix defined in the DDL. In our example, the prefix is "ABC". The `nOrderInDBD` is the record number in the database definition (.DBD) file. When DDLP compiles the DDL into DBD format, record tables are stored describing each record structure definition. The record tables are stored in alphabetical order in the DBD file. In our example, the record order in the DBD is 'address', 'client', then 'setup'. Therefore, when the first 'client' record is added to the database, ABC0001.DAT and ABC0001.KEY are created. When the first 'address' record is added, ABC0000.DAT and ABC0000.KEY are created.

If a record definition does not contain a key field, a key file is not created for this record.

Key File Layout

Key files are organized as a series of pages. Pages contain a series of slots. The slots contain the key data. Slots on a page are in sorted order. A key file will contain pages of keys for a specific record type. For example, in SAMPLEDB.DDL, the 'client' key file will contain pages of keys for the 'client' record. Some pages will contain client numbers and some pages will contain client names. A page will not contain both client numbers and client names.

The key file structures are listed below:

```

struct key_file_index
{
    CHAR          name[12];          /* Key file name */
    UINT          nextavailpage;    /* Next available page */
    UINT          firstdelpage;     /* First page in the
        delete */
                                /* chain. */
    UINT          pagenbr[MAXKEY];  /* Key 1st page index */
};

```

This structure is included at the beginning of every key file. It contains necessary pointers for finding the next available page, first deleted page in the delete chain, and the first page for each key field defined in the record.

```

struct key_page_index
{
    UINT                prevpage;                /* Previous page in sort
        tree */
    UINT                nextpage;                /* Next page in sort tree
        */
    UINT                slotsused;                /* Number of slots used
        on page */
    UINT                slotsize;                /* Size of key slot */
    UINT                flags;                    /* Bit 0 - page is full */
};

```

This structure is included at the beginning of each page in the key file. The slots in the key file consist of nothing but raw key field data.

Data File Layout

Data files do not contain pages. They are organized as a series of slots in a file. Pages are not needed here because data files contain only record data. They are indexed by their respective key files.

The data file structures are listed below:

```

struct data_file_index
{
    CHAR                name[12];                /* Data file name */
    ULONG               nextavailslot;          /* Next available slot */
    ULONG               firstdelslot;          /* First slot in the delete
        */
                                                /* chain. */
    UINT                slotsize;                /* Size of data slot */
    CHAR                filler[10];
};

```

This structure is included at the beginning of every data file. It contains necessary pointers for finding the next available slot, first deleted slot in the delete chain, and the data slot size.

```

struct data_slot_index
{
    UINT                offsetto data;          /* Offset to actual data
        */
    ULONG               nextdel;                /* DBA of next member
        in the */
                                                /* delete chain. */
};

```

This structure is included at the beginning of each data slot. A data slot also contains owner and/or member data if the record type is an owner of or member of another record. The owner and member pointer tables are not shown here. In summary, a data slot contains a data_slot_index, owner data tables, member data table, followed by the actual data record.

Modifying Internal Definitions

It is relatively easy to change some of the global definitions used by the CDB For Windows database server. In some extreme cases, modification may be necessary. This, of course, depends on your database model (data definition file). Source code is required to make any of the changes to the definitions listed below.

All definitions described are included in DBMGR.H.

#define NBRHANDLES 8

This value is the number of database files the database server, DBSERVER.EXE, can have open at one time. A data file exists for every record defined in the DDL if at least one record of that type has been added. If the DDL structure definition contains one or more key fields, a key file will also be created.

```
struct client
{
    key long          lClientNbr;
    key char          szName[31];
    char             szDescription[61];
    double           dBalance;
};
```

In this DDL example, two files will be created when the first record of this type is added to the database. A data file will be created and because at least 1 key field exists, a key file will also be created.

The database server uses an LRU (least recently used) algorithm to manage database file handles. If the server needs to open a file and 8 database files are already open, the server closes the least recently used handle and proceeds to open the new file. The new file handle is then placed in the LRU table.

If your database model contains more than 8 database files, database performance may be enhanced by increasing the NBRHANDLES value. Note: The maximum number of file handles available for a single task under DOS is 20. 5 are reserved for internal use.

#define MAXKEY 8

This value is the maximum number of key fields that a single record definition can contain. Increase this value only if you have more than 8 key fields defined in a single record definition.

#define KEYPAGESIZE 512

Key fields are stored in sorted order in slots on pages. A key file is made up of a header and a series of these pages. KEYPAGESIZE is the size of a key page. If your database key fields are very large, you might increase the performance of the database by increasing this value. If modified, KEYPAGESIZE should be a multiple of the average key field length. Note: The larger the key page size, the longer the access time for reads and writes.

#define NBRPAGES 16

This value is the number of key pages that are buffered in RAM by the database server. These buffers are managed using an LRU (least recently used) algorithm for maximum efficiency.

#define DATAPAGESIZE 2048

Data records are stored in slots on pages. The pages are stored in the data file (.DAT). This value is the size of the data pages. It is recommended that this value be a power of 2.

#define DATASLOTSIZE 1024

This value is the maximum size of a data slot. A data slot contains a small header, followed by owner tables (if any), followed by member tables (if any), followed by the actual data. You will need to increase this value if your record sizes, when plugged into the formula below, exceed 1024.

Formula:

6 +
(number of members this record owns * 8) +
(number of owners that own this record * 12) +
C structure length (in bytes)

If you have very large C structures you should check them. It is recommended that DATASLOTSIZE be a power of 2.

Utilities

DDL.P.EXE

DDL.P is the Data Definition Language Parser (compiler). It reads the DDL file and creates a binary database definition file with a .DBD extension. DDL.P also creates a C header file with a .H extension.

The DBD file name is used with the DbOpen function call. The DbOpen function passes the DBD file name as a parameter. The DBD file is read into memory by the database server (DBSERVER.EXE) and serves as a roadmap for the database.

The maximum size of a .DDL file that DDL.P can process is 65535 bytes. A complete list of DDL.P error messages are provided in the 'Error Messages' section in this manual.

Syntax:

DDL.P filename(.ddl)

Example:

DDL.P sampledb.ddl

In this example, DDL.P will create SAMPLEDB.DBD and SAMPLEDB.H if the compilation is successful.

DBDLIST.EXE

DBDLIST displays the contents of the binary database definition file (.DBD) created by DDL.P. A header, record definitions, owner definitions, member definitions, and field definitions are displayed.

DBDLIST does not display the contents of any data or key files.

Syntax:

DBDLIST filename.dbd

Example:

DBDLIST sampledb.dbd

Using the C-API

Introduction

The CDB For Windows C-API library is callable from both C and C++ modules. Over 40 functions are available. Function prototypes are defined in DBMGR.H.

Functions that have a prefix of 'X' are extended functions. An extended function is a superset of an existing function, e.g. XDbRecordAdd(...). The extended functions were created for programmer convenience.

Functions by Category

Database Management

DbClose	Close a database.
DbFlush	Flush all data files to disk.
DbOpen	Open a database.

Record Management

DbRecordAdd	Add a record.
DbRecordDelete	Delete a record.
DbRecordUpdate	Update a record.
XDbRecordAdd	Extended function - Add a record.
XDbRecordUpdate	Extended function - Update a record.

Record Find

DbRecordFindByKey	Find a record by key value.
DbRecordFindFirst	Find the first record.
DbRecordFindLast	Find the last record.
DbRecordFindNext	Find the next record.
DbRecordFindPrev	Find the previous record.
XDbRecordFindByKey	Extended function - Find a record by key value.

Record Retrieval

DbRecordGetByKey	Get a record by key value.
DbRecordGetCurrent	Get the current record.
DbRecordGetFirst	Get the first record.
DbRecordGetLast	Get the last record.
DbRecordGetNext	Get the next record.
DbRecordGetPrev	Get the previous record.
XDbRecordGetByKey	Extended function - Get a record by key value.
XDbRecordGetCurrent	Extended function - Get the current record.
XDbRecordGetFirst	Extended function - Get the first record.
XDbRecordGetLast	Extended function - Get the last record.
XDbRecordGetNext	Extended function - Get the next record.
XDbRecordGetPrev	Extended function - Get the previous record.

Record Currency

DbRecordGetCurrency	Get the currency table of a record type.
DbRecordUpdCurrency	Update the currency table of a record type.
XDbRecordGetCurrency	Extended function - Get the currency table of a record type.
XDbRecordUpdCurrency	Extended function - Update the currency table of a record type.

Set Management

DbSetAdd	Make a set connection between two records.
DbSetDelete	Remove a set connection between two records.

Set Find

DbSetFindFirst	Find the first member record in an owner/member relationship.
DbSetFindLast	Find the last member record in an owner/member relationship.
DbSetFindNext	Find the next member record in an owner/member relationship.
DbSetFindPrev	Find the previous member record in an owner/member relationship.

Set Retrieval

DbSetGetFirst	Get the first member record in an owner/member relationship.
DbSetGetLast	Get the last member record in an owner/member relationship.
DbSetGetNext	Get the next member record in an owner/member relationship.
DbSetGetOwner	Get the owner record of a member in an owner/member relationship.
DbSetGetPrev	Get the previous member record in an owner/member relationship.
XDbSetGetFirst	Extended function - Get the first member record in an owner/member relationship.
XDbSetGetLast	Extended function - Get the last member record in an owner/member relationship.
XDbSetGetNext	Extended function - Get the next member record in an owner/member relationship.
XDbSetGetOwner	Extended function - Get the owner record of a member in an owner/member relationship.
XDbSetGetPrev	Extended function - Get the previous member record in an owner/member relationship.

DbClose

Summary

DWORD FAR PASCAL DbClose(HANDLE hDb);

Parameters

hDb **HANDLE** Identifies the database to be closed.

Description

The DbClose function closes an open database. All database files are closed, memory deallocated, and the associated server instance, DBSERVER.EXE, is terminated.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC or E_DOS. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HWND hWnd)
{
    HANDLE hDb;
    DWORD dwStatus;

    if( dwStatus = DbOpen( hWnd, ".\\", "test.dbd", &hDb))
    {
        /* Database not opened */
    }

    /* Other CDB calls... */

    if( dwStatus = DbClose( hDb))
    {
        /* Database not closed */
    }
}
```

DbFlush

Summary

DWORD FAR PASCAL DbFlush(HANDLE hDb);

Parameters

hDb **HANDLE** Identifies the database to be flushed.

Description

The DbFlush function forces all data written to the database to disk. If DOS 3.30 or higher is being used, DbFlush uses the DOS Commit function call. Otherwise, the open files are closed and then reopened.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC or E_DOS. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb, struct client far *lpClient)
{
    DWORD dwStatus;

    if( dwStatus = XDbRecordUpdate( hDb, "client", lpClient, sizeof( struct client)) )
    {
        /* Record not updated */
    }

    if( dwStatus = DbFlush( hDb))
    {
        /* Database not flushed */
    }
}
```

DbOpen

Summary

DWORD FAR PASCAL DbOpen(HWND hParentWnd, LPSTR szDbDir, LPSTR szDbName, HANDLE *hDb)

Parameters

<i>hParentWnd</i>	HWND Identifies the parent window of the database.
<i>szDbDir</i>	LPSTR Identifies the directory where CDB will attempt to open the .DBD (Database Definition) file. CDB will also attempt to open and/or create all associated database files in this directory. If NULL, CDB will use the current directory. Note: If a directory name is present, it must end with a backslash. e.g. "C:\\PRODUCTA\\".
<i>szDbName</i>	LPSTR Identifies the .DBD (Database Definition) file.
<i>hDb</i>	HANDLE * Pointer to the storage location of the database handle. If DbOpen is successful, this location will contain the handle for the opened database.

Description

The DbOpen function opens a CDB database. The database definition file (*szDbName*) is created by DDLP.EXE. Multiple databases can be opened concurrently by a single MS-Windows application. The handle returned, *hDb*, is to be used in subsequent calls to the database.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_ALREADYOPEN, E_LOADMODULE, or E_DOS. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HWND hWnd)
{
    HANDLE hDb;
    DWORD dwStatus;

    if( dwStatus = DbOpen( hWnd, "C:\\PRODUCTA\\", "test.dbd", &hDb))
    {
        /* Error opening database */
    }

    /* Other CDB calls... */
}
```

DbRecordAdd

Summary

DWORD FAR PASCAL DbRecordAdd(HANDLE hDb, LPSTR szRecName, HANDLE hData)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

hData **HANDLE** Identifies the record data to be added. This handle must be allocated using the GMEM_DDESHARE flag.

Description

The DbRecordAdd function adds a record to the database.

See Also

XDbRecordAdd

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, or E_NORECNAME. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

DWORD Function( HANDLE hDb, HANDLE hClientData)
{
    DWORD dwStatus;

    if( dwStatus = DbRecordAdd( hDb, "client", hClientData))
    {
        /* Error adding record */
    }

    return dwStatus;
}
```

DbRecordDelete

Summary

DWORD FAR PASCAL DbRecordDelete(HANDLE hDb, LPSTR szRecName)

Parameters

hDb **HANDLE** Identifies the database.
szRecName **LPSTR** Pointer to the record name.

Description

The DbRecordDelete function deletes a record from the database. The record deleted is the current record of the *szRecName* type.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECFNAME, or E_NOCURRENT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    if( dwStatus = DbRecordDelete( hDb, "client"))
    {
        /* Error deleting record */
    }
}
```

DbRecordFindByKey

Summary

DWORD FAR PASCAL DbRecordFindByKey(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, HANDLE hKey)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

hKey **HANDLE** Identifies the key data to be used for the record search. This handle must be allocated using the GMEM_DDESHARE flag.

Description

The DbRecordFindByKey function searches for a specific record using a key field and key value.

See Also

XDbRecordFindByKey

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, E_NOTFOUND, or E_NEXTGUESS. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hKey;
    LONG FAR *lpKey;
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Store the key data */
    hKey = GlobalAlloc( GMEM_MOVEABLE | GMEM_ZEROINIT, (DWORD)sizeof(LONG));
    lpKey = (LONG FAR *)GlobalLock( hKey);
    *lpKey = IClientNbr;
    GlobalUnlock( hKey);

    if( dwStatus = DbRecordFindByKey( hDb, "client", "IClientNbr", hKey))
    {
        /* Record not found */
    }

    GlobalFree( hKey);
}
```

DbRecordFindFirst

Summary

DWORD FAR PASCAL DbRecordFindFirst(HANDLE hDb, LPSTR szRecName, LPSTR szFldName)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

Description

The DbRecordFindFirst function sets the database currency to the first logical record sorted by *szFldName*. For more on currency, see the Database Currency section in this manual.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;

    /* Delete all client records in database */
    while( ! DbRecordFindFirst( hDb, "client", "IClientNbr"))
    {
        if( dwStatus = DbRecordDelete( hDb, "client"))
        {
            /* Error deleting record */
        }
    }
}
```

DbRecordFindLast

Summary

DWORD FAR PASCAL DbRecordFindLast(HANDLE hDb, LPSTR szRecName, LPSTR szFldName)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

Description

The DbRecordFindLast function sets the database currency to the last logical record sorted by *szFldName*. For more on currency, see the Database Currency section in this manual.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;

    /* Delete all client records in database */
    while( ! DbRecordFindLast( hDb, "client", "IClientNbr"))
    {
        if( dwStatus = DbRecordDelete( hDb, "client"))
        {
            /* Error deleting record */
        }
    }
}
```

DbRecordFindNext

Summary

DWORD FAR PASCAL DbRecordFindNext(HANDLE hDb, LPSTR szRecName, LPSTR szFldName)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

Description

The DbRecordFindNext function sets the database currency to the next logical record sorted by *szFldName*. The record must have currency before this call is executed. For more on currency, see the Database Currency section in this manual.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEKEY, E_NOTFOUND, E_NOCURRENT, or E_NONEXT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

DWORD Function( HANDLE hDb)
{
    /* Down arrow key pressed, check for next */
    /* record in database. */
    return( DbRecordFindNext( hDb, "client", "IClientNbr"));
}
```

DbRecordFindPrev

Summary

DWORD FAR PASCAL DbRecordFindPrev(HANDLE hDb, LPSTR szRecName, LPSTR szFldName)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

Description

The DbRecordFindPrev function sets the database currency to the previous logical record sorted by *szFldName*. The record must have currency before this call is executed. For more on currency, see the Database Currency section in this manual.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEKEY, E_NOTFOUND, E_NOCURRENT, or E_NOPREV. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

DWORD Function( HANDLE hDb)
{
    /* Up arrow key pressed, check for previous */
    /* record in database. */
    return( DbRecordFindPrev( hDb, "client", "IClientNbr"));
}
```

DbRecordGetByKey

Summary

DWORD FAR PASCAL DbRecordGetByKey(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, HANDLE hTarget, HANDLE hKey)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szRecName</i>	LPSTR Pointer to the record name.
<i>szFldName</i>	LPSTR Pointer to the field name. Must be a key field.
<i>hTarget</i>	HANDLE Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.
<i>hKey</i>	HANDLE Identifies the key data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbRecordGetByKey function retrieves a record using a key value. If the exact match cannot be found the function will return E_NEXTGUESS specifying that the data returned is the next best guess.

See Also

XDbRecordGetByKey

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, E_NOTFOUND, or E_NEXTGUESS. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hKey, hTarget;
    LONG FAR *lpKey;
    DWORD dwStatus;

    /* Store the key data (1000L) */
    hKey = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof(LONG));
    lpKey = (LONG FAR *)GlobalLock( hKey);
    *lpKey = 1000L;
    GlobalUnlock( hKey);

    /* Allocate the target area */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof(struct
        client));

    /* Find client record with client number equal 1000L */
    if( dwStatus = DbRecordGetByKey( hDb, "client", "IClientNbr", hTarget, hKey))
    {
        /* Client not retrieved */
    }

    /* If successful, record returned in 'hTarget' */
}
```

DbRecordGetCurrency

Summary

DWORD FAR PASCAL DbRecordGetCurrency(HANDLE hDb, LPSTR szRecName, HANDLE hTarget)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

hTarget **HANDLE** Identifies the storage area for the currency information. Must be allocated using GMEM_DDESHARE flag.

Description

The DbRecordGetCurrency function retrieves the current currency table for a specific record. For more on currency, see the Database Currency section in this manual.

See Also

XDbRecordGetCurrency

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, or E_NORECNAME. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hCurrency;
    DWORD dwStatus;

    /* Allocate the target area for currency table. */
    /* Note: 'currency_index is defined in DBMGR.H */
    hCurrency = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof(struct
        currency_index));

    /* Get the currency for the client record */
    if( dwStatus = DbRecordGetCurrency( hDb, "client", hCurrency))
    {
        /* Currency not retrieved */
    }

    /* Other processing goes here... */

    /* Restore the currency for the client record */
    if( dwStatus = DbRecordUpdCurrency( hDb, "client", hCurrency))
    {
        /* Currency not updated */
    }
}
```

DbRecordGetCurrent

Summary

DWORD FAR PASCAL DbRecordGetCurrent(HANDLE hDb, LPSTR szRecName, HANDLE hTarget)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

hTarget **HANDLE** Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbRecordGetCurrent function retrieves the record that has currency (or 'is current') for that record type (record name). Each record type has its own currency table. For more on currency, see the Database Currency section in this manual.

See Also

XDbRecordGetCurrent

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, or E_NOCURRENT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Check for client #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof(LONG)))
    {
        /* Record not found */
    }

    /* Allocate storage for the record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        client));

    /* Retrieve it */
    if( dwStatus = DbRecordGetCurrent( hDb, "client", hTarget))
    {
        /* Record not retrieved */
    }
    /* If successful, record returned in 'hTarget' */
}
```

DbRecordGetFirst

Summary

DWORD FAR PASCAL DbRecordGetFirst(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, HANDLE hTarget)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

hTarget **HANDLE** Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbRecordGetFirst function retrieves the first record by the key field passed. After this call, the currency for this record type is set to the first record.

See Also

XDbRecordGetFirst

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;

    /* Allocate space for the client record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        client));

    /* Get the first record sorted by client number*/
    if( dwStatus = DbRecordGetFirst( hDb, "client", "IClientNbr", hTarget))
    {
        /* Record not retrieved */
    }
    /* If successful, record returned in 'hTarget' */
}
```

DbRecordGetLast

Summary

DWORD FAR PASCAL DbRecordGetLast(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, HANDLE hTarget)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

hTarget **HANDLE** Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbRecordGetLast function retrieves the last record by the key field passed. After this call, the currency for this record type is set to the last record.

See Also

XDbRecordGetLast

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;

    /* Allocate space for the client record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        client));

    /* Get the last record sorted by client number*/
    if( dwStatus = DbRecordGetLast( hDb, "client", "IClientNbr", hTarget))
    {
        /* Record not retrieved */
    }
    /* If successful, record returned in 'hTarget' */
}
```

DbRecordGetNext

Summary

DWORD FAR PASCAL DbRecordGetNext(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, HANDLE hTarget)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

hTarget **HANDLE** Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbRecordGetNext function retrieves the next record by the key field passed. After this call, the currency for this record type is set to the record retrieved.

See Also

XDbRecordGetNext

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, E_NOTFOUND, E_NOCURRENT, or E_NONEXT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;

    /* Allocate space for the client record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        client));

    /* Get the next record sorted by client number*/
    if( dwStatus = DbRecordGetNext( hDb, "client", "IClientNbr", hTarget))
    {
        /* Record not retrieved */
    }
    /* If successful, record returned in 'hTarget' */
}
```

DbRecordGetPrev

Summary

DWORD FAR PASCAL DbRecordGetPrev(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, HANDLE hTarget)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szRecName</i>	LPSTR Pointer to the record name.
<i>szFldName</i>	LPSTR Pointer to the field name. Must be a key field.
<i>hTarget</i>	HANDLE Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbRecordGetPrev function retrieves the previous record by the key field passed. After this call, the currency for this record type is set to the record retrieved.

See Also

XDbRecordGetPrev

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, E_NOTFOUND, E_NOCURRENT, or E_NOPREV. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;

    /* Allocate space for the client record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        client));

    /* Get the previous record sorted by client number*/
    if( dwStatus = DbRecordGetPrev( hDb, "client", "IClientNbr", hTarget))
    {
        /* Record not retrieved */
    }
    /* If successful, record returned in 'hTarget' */
}
```

DbRecordUpdate

Summary

DWORD FAR PASCAL DbRecordUpdate(HANDLE hDb, LPSTR szRecName, HANDLE hData)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

hData **HANDLE** Identifies the updated record data. This handle must be allocated using the GMEM_DDESHARE flag.

Description

The DbRecordUpdate function updates a database record. The record to be updated must be current. For more on currency, see the Database Currency section in this manual.

See Also

XDbRecordUpdate

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, or E_NOCURRENT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hData;
    DWORD dwStatus;

    /* Allocate space for the client record */
    hData = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        client));

    /* Get the first record */
    if( dwStatus = DbRecordGetFirst( hDb, "client", "IClientNbr", hData))
    {
        /* Record not retrieved */
    }

    /* Modify input logic goes here... */

    /* Update the record */
    if( dwStatus = DbRecordUpdate( hDb, "client", "IClientNbr", hData))
    {
        /* Error updating record */
    }
}
```

DbRecordUpdCurrency

Summary

DWORD FAR PASCAL DbRecordUpdCurrency(HANDLE hDb, LPSTR szRecName, HANDLE hData)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

hData **HANDLE** Identifies the storage area for the currency information. Must be allocated using GMEM_DDESHARE flag.

Description

The DbRecordUpdCurrency function updates the currency for a specific record type. For more on currency, see the Database Currency section in this manual.

See Also

XDbRecordUpdCurrency

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, or E_NORECNAME. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hCurrency;
    DWORD dwStatus;

    /* Allocate the target area for currency table. */
    /* Note: 'currency_index' is defined in DBMGR.H */
    hCurrency = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof(struct
        currency_index));

    /* Get the currency for the client record */
    if( dwStatus = DbRecordGetCurrency( hDb, "client", hCurrency))
    {
        /* Error retrieving currency */
    }

    /* Other processing goes here... */

    /* Restore the currency for the client record */
    if( dwStatus = DbRecordUpdCurrency( hDb, "client", hCurrency))
    {
        /* Error updating currency */
    }
}
```

DbSetAdd

Summary

DWORD FAR PASCAL DbSetAdd(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

Description

The DbSetAdd function makes a set connection between two records. Both records must have currency before making the call.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, or E_NOCURRENT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;

    /* Add an client record */
    /* Assuming 'sclient' is global and structure prefilled */
    if( dwStatus = XDbRecordAdd( hDb, "client", &sclient, sizeof(struct client)))
    {
        /* Error adding record */
    }

    /* Add an address record */
    /* Assuming 'saddress' is global and structure prefilled */
    if( dwStatus = XDbRecordAdd( hDb, "address", &saddress, sizeof(struct address)))
    {
        /* Error adding record */
    }

    /* Make a set connection between records */
    /* After this call, the 'client' record is the owner of the 'address' record */
    /* The 'address' record is a member of the 'client' record */
    if( dwStatus = DbSetAdd( hDb, "client", "address"))
    {
        /* Error making set connection */
    }
}
```

DbSetDelete

Summary

DWORD FAR PASCAL DbSetDelete(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

Description

The DbSetDelete function removes a set connection between two records. Both records must have currency before making the call. This function does not delete either record, it only removes the connection between the two.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, or E_NOCURRENT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Set the currency for the address record to the first member of the client record */
    if( dwStatus = DbSetFindFirst( hDb, "client", "address"))
    {
        /* First member not found */
    }

    /* Delete the owner/member set connection */
    if( dwStatus = DbSetDelete( hDb, "client", "address"))
    {
        /* Error deleting record */
    }
}
```

DbSetFindFirst

Summary

DWORD FAR PASCAL DbSetFindFirst(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

Description

The DbSetFindFirst function sets the database currency for the member record to the first member in the owner/member set relation. The owner record must have currency before this call. For more on currency, see the Database Currency section in this manual.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Set the currency for the address record to the first member of the client record */
    if( dwStatus = DbSetFindFirst( hDb, "client", "address"))
    {
        /* First member not found */
    }
}
```

DbSetFindLast

Summary

DWORD FAR PASCAL DbSetFindLast(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

Description

The DbSetFindLast function sets the database currency for the member record to the last member in the owner/member set relation. The owner record must have currency before this call. For more on currency, see the Database Currency section in this manual.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Set the currency for the address record to the last member of the client record */
    if( dwStatus = DbSetFindLast( hDb, "client", "address"))
    {
        /* Last member not found */
    }
}
```

DbSetFindNext

Summary

DWORD FAR PASCAL DbSetFindNext(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

Description

The DbSetFindNext function sets the database currency for the member record to the next member in the owner/member set relation. Both owner and member records must have currency before this call. For more on currency, see the Database Currency section in this manual.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NONEXT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Set the currency for the address record to the first member of the client record */
    if( dwStatus = DbSetFindFirst( hDb, "client", "address"))
    {
        /* First member not found */
    }

    /* Set the currency for the address record to the next member of the client record */
    if( dwStatus = DbSetFindNext( hDb, "client", "address"))
    {
        /* Next member not found */
    }
}
```

DbSetFindPrev

Summary

DWORD FAR PASCAL DbSetFindPrev(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

Description

The DbSetFindPrev function sets the database currency for the member record to the previous member in the owner/member set relation. Both owner and member records must have currency before this call. For more on currency, see the Database Currency section in this manual.

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOPREV. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Set the currency for the address record to the last member of the client record */
    if( dwStatus = DbSetFindLast( hDb, "client", "address"))
    {
        /* Last member not found */
    }

    /* Set the currency for the address record to the previous member of the client record */
    if( dwStatus = DbSetFindPrev( hDb, "client", "address"))
    {
        /* Previous member not found */
    }
}
```

DbSetGetFirst

Summary

DWORD FAR PASCAL DbSetGetFirst(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, HANDLE hTarget)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szOwnerName</i>	LPSTR Pointer to the owner record name.
<i>szMemberName</i>	LPSTR Pointer to the member record name.
<i>hTarget</i>	HANDLE Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbSetGetFirst function retrieves the first member of an owner/member set relation. The owner record must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

XDbSetGetFirst

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Allocate storage for the address record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        address));

    /* Get the first address record */
    if( dwStatus = DbSetGetFirst( hDb, "client", "address", hTarget))
    {
        /* First member not found */
    }
}
```

DbSetGetLast

Summary

DWORD FAR PASCAL DbSetGetLast(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, HANDLE hTarget)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szOwnerName</i>	LPSTR Pointer to the owner record name.
<i>szMemberName</i>	LPSTR Pointer to the member record name.
<i>hTarget</i>	HANDLE Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbSetGetLast function retrieves the last member of an owner/member set relation. The owner record must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

XDbSetGetLast

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Allocate storage for the address record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        address));

    /* Get the last address record */
    if( dwStatus = DbSetGetLast( hDb, "client", "address", hTarget))
    {
        /* Last member not found */
    }
}
```

DbSetGetNext

Summary

DWORD FAR PASCAL DbSetGetNext(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, HANDLE hTarget)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szOwnerName</i>	LPSTR Pointer to the owner record name.
<i>szMemberName</i>	LPSTR Pointer to the member record name.
<i>hTarget</i>	HANDLE Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbSetGetNext function retrieves the next member of an owner/member set relation. Both owner and member records must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

XDbSetGetNext

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NONEXT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Allocate storage for the address record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        address));

    /* Get the first address record */
    if( dwStatus = DbSetGetFirst( hDb, "client", "address", hTarget))
    {
        /* First member not found */
    }

    /* Get the next address record */
    if( dwStatus = DbSetGetNext( hDb, "client", "address", hTarget))
    {
        /* Next member not found */
    }
}
```

DbSetGetOwner

Summary

DWORD FAR PASCAL DbSetGetOwner(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, HANDLE hTarget)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szOwnerName</i>	LPSTR Pointer to the owner record name.
<i>szMemberName</i>	LPSTR Pointer to the member record name.
<i>hTarget</i>	HANDLE Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbSetGetOwner function retrieves the owner record of a member record set relation. The member record must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

XDbSetGetOwner

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOOWNER. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;
    DWORD dwStatus;

    /* Set the currency to the first invoice record */
    if( dwStatus = DbRecordFindFirst( hDb, "invoice", "IInvoiceNbr"))
    {
        /* Invoice not found */
    }

    /* Allocate storage for the client record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
    client));

    /* Get the client record for this invoice */
    if( dwStatus = DbSetGetOwner( hDb, "client", "invoice", hTarget))
    {
        /* Client record not found */
    }
}
```

DbSetGetPrev

Summary

DWORD FAR PASCAL DbSetGetPrev(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, HANDLE hTarget)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szOwnerName</i>	LPSTR Pointer to the owner record name.
<i>szMemberName</i>	LPSTR Pointer to the member record name.
<i>hTarget</i>	HANDLE Identifies the storage area for the record data. Must be allocated using GMEM_DDESHARE flag.

Description

The DbSetGetPrev function retrieves the previous member of an owner/member set relation. Both owner and member records must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

XDbSetGetPrev

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOPREV. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    HANDLE hTarget;
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Allocate storage for the address record */
    hTarget = GlobalAlloc( GMEM_MOVEABLE | GMEM_DDESHARE, (DWORD)sizeof( struct
        address));

    /* Get the last address record */
    if( dwStatus = DbSetGetLast( hDb, "client", "address", hTarget))
    {
        /* Last member not found */
    }

    /* Get the previous address record */
    if( dwStatus = DbSetGetPrev( hDb, "client", "address", hTarget))
    {
        /* Previous member not found */
    }
}
```

XDbRecordAdd

Summary

DWORD FAR PASCAL XDbRecordAdd(HANDLE hDb, LPSTR szRecName, LPVOID lpData, short nDataLen)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szRecName</i>	LPSTR Pointer to the record name.
<i>lpData</i>	LPVOID Pointer to the data to be added.
<i>nDataLen</i>	short Length of the data in <i>lpData</i> .

Description

The XDbRecordAdd function adds a record to the database.

See Also

DbRecordAdd

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, or E_NORECNAME. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

DWORD Function( HANDLE hDb, struct client far *lpClient)
{
    DWORD dwStatus;

    if( dwStatus = XDbRecordAdd( hDb, "client", lpClient, sizeof( struct client)))
    {
        /* Record not added */
    }

    return dwStatus;
}
```

XDbRecordFindByKey

Summary

DWORD FAR PASCAL XDbRecordFindByKey(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, LPVOID lpKey, short nKeyLen)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szRecName</i>	LPSTR Pointer to the record name.
<i>szFldName</i>	LPSTR Pointer to the field name. Must be a key field.
<i>lpKey</i>	LPVOID Pointer to key data to be used for find.
<i>nKeyLen</i>	short Length of the key data in <i>lpKey</i> .

Description

The XDbRecordFindByKey function searches for a specific record using a key field and key value.

See Also

DbRecordFindByKey

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, E_NOTFOUND, or E_NEXTGUESS. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;

    /* Set the currency for the client record type to client #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Client #1000 not found*/
    }
}
```

XDbRecordGetByKey

Summary

DWORD FAR PASCAL XDbRecordGetByKey(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, LPVOID lpTarget, short nTargetLen, LPVOID lpKey, short nKeyLen)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szRecName</i>	LPSTR Pointer to the record name.
<i>szFldName</i>	LPSTR Pointer to the field name. Must be a key field.
<i>lpTarget</i>	LPVOID Pointer to the storage location for the retrieved data.
<i>nTargetLen</i>	short Length of storage location <i>lpTarget</i> .
<i>lpKey</i>	LPVOID Pointer to key data to be used for search.
<i>nKeyLen</i>	short Length of the key data in <i>lpKey</i> .

Description

The XDbRecordGetByKey function retrieves a record using a key value. If the exact match cannot be found the function will return E_NEXTGUESS specifying that the data returned is the next best guess.

See Also

DbRecordGetByKey

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEKEY, E_NOTFOUND, or E_NEXTGUESS. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;
    struct client sclient;

    /* Get client record #1000 */
    if( dwStatus = XDbRecordGetByKey( hDb, "client", "IClientNbr", &sclient, sizeof( struct client),
        &IClientNbr, sizeof(LONG)))
    {
        /* Record not found, if dwStatus == E_NEXTGUESS, 'sclient' contains the next best
        guess */
    }
}
```

XDbRecordGetCurrency

Summary

DWORD FAR PASCAL XDbRecordGetCurrency(HANDLE hDb, LPSTR szRecName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

lpTarget **LPVOID** Pointer to the storage location for the retrieved currency data.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbRecordGetCurrency function retrieves the current currency table for a specific record. For more on currency, see the Database Currency section in this manual.

See Also

DbRecordGetCurrency

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, or E_NORECNAME. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;
    /* Note: 'currency_index' is defined in DBMGR.H */
    struct currency_index scurrency;

    /* Get the currency for the client record */
    if( dwStatus = XDbRecordGetCurrency( hDb, "client", &scurrency, sizeof( struct
        currency_index)))
    {
        /* Currency not retrieved */
    }

    /* Other processing goes here... */

    /* Restore the currency for the client record */
    if( dwStatus = XDbRecordUpdCurrency( hDb, "client", &scurrency, sizeof( struct
        currency_index)))
    {
        /* Currency not updated */
    }
}
```

XDbRecordGetCurrent

Summary

DWORD FAR PASCAL XDbRecordGetCurrent(HANDLE hDb, LPSTR szRecName, LPVOID lpTarget, short nTargetLen)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szRecName</i>	LPSTR Pointer to the record name.
<i>lpTarget</i>	LPVOID Pointer to the storage location for the retrieved record.
<i>nTargetLen</i>	short Length of storage location <i>lpTarget</i> .

Description

The XDbRecordGetCurrent function retrieves the record that has currency (or 'is current') for that record type (record name). Each record type has its own currency table. For more on currency, see the Database Currency section in this manual.

See Also

DbRecordGetCurrent

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, or E_NOCURRENT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;
    struct client sclient;

    /* Check for client #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof(LONG)))
    {
        /* Client not found */
    }

    /* Retrieve it */
    if( dwStatus = XDbRecordGetCurrent( hDb, "client", &sclient, sizeof( struct client)))
    {
        /* Error retrieving record */
    }
}
```

XDbRecordGetFirst

Summary

DWORD FAR PASCAL XDbRecordGetFirst(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

lpTarget **LPVOID** Pointer to the storage location for the retrieved record.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbRecordGetFirst function retrieves the first record by the key field passed. After this call, the currency for this record type is set to the first record.

See Also

DbRecordGetFirst

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;
    struct client sclient;

    /* Get the first record sorted by client number*/
    if( dwStatus = XDbRecordGetFirst( hDb, "client", "lClientNbr", &sclient, sizeof( struct client)))
    {
        /* Record not found */
    }
}
```

XDbRecordGetLast

Summary

DWORD FAR PASCAL XDbRecordGetLast(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

lpTarget **LPVOID** Pointer to the storage location for the retrieved record.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbRecordGetLast function retrieves the last record by the key field passed. After this call, the currency for this record type is set to the last record.

See Also

DbRecordGetLast

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;
    struct client sclient;

    /* Get the last record sorted by client number*/
    if( dwStatus = XDbRecordGetLast( hDb, "client", "IClientNbr", &sclient, sizeof( struct client)))
    {
        /* Record not found */
    }
}
```

XDbRecordGetNext

Summary

DWORD FAR PASCAL XDbRecordGetNext(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

lpTarget **LPVOID** Pointer to the storage location for the retrieved record.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbRecordGetNext function retrieves the next record by the key field passed. After this call, the currency for this record type is set to the record retrieved.

See Also

DbRecordGetNext

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, E_NOTFOUND, E_NOCURRENT, or E_NONEXT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;
    struct client sclient;

    /* Get the next record sorted by client number*/
    if( dwStatus = XDbRecordGetNext( hDb, "client", "IClientNbr", &sclient, sizeof( struct client)))
    {
        /* Record not found */
    }
}
```

XDbRecordGetPrev

Summary

DWORD FAR PASCAL XDbRecordGetPrev(HANDLE hDb, LPSTR szRecName, LPSTR szFldName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

szFldName **LPSTR** Pointer to the field name. Must be a key field.

lpTarget **LPVOID** Pointer to the storage location for the retrieved record.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbRecordGetPrev function retrieves the previous record by the key field passed. After this call, the currency for this record type is set to the record retrieved.

See Also

DbRecordGetPrev

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_NOFLDNAME, E_NOTAKEY, E_NOTFOUND, E_NOCURRENT, or E_NOPREV. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;
    struct client sclient;

    /* Get the previous record sorted by client number*/
    if( dwStatus = XDbRecordGetPrev( hDb, "client", "IClientNbr", &sclient, sizeof( struct client)))
    {
        /* Record not found */
    }
}
```

XDbRecordUpdate

Summary

DWORD FAR PASCAL XDbRecordUpdate(HANDLE hDb, LPSTR szRecName, LPVOID lpData, short nDataLen)

Parameters

hDb **HANDLE** Identifies the database.

szRecName **LPSTR** Pointer to the record name.

lpData **LPVOID** Pointer to the storage location of the updated record.

nDataLen **short** Length of storage location *lpData*.

Description

The XDbRecordUpdate function updates a database record. The record to be updated must be current. For more on currency, see the Database Currency section in this manual.

See Also

DbRecordUpdate

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, or E_NOCURRENT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;
    struct client sclient;

    /* Get the first record */
    if( dwStatus = XDbRecordGetFirst( hDb, "client", "IClientNbr", &sclient, sizeof( struct client)))
    {
        /* Record not found */
    }

    /* Modify input logic goes here... */

    /* Update the record */
    if( dwStatus = XDbRecordUpdate( hDb, "client", "IClientNbr", &sclient, sizeof( struct client)))
    {
        /* Record not updated */
    }
}
```

XDbRecordUpdCurrency

Summary

DWORD FAR PASCAL XDbRecordUpdCurrency(HANDLE hDb, LPSTR szRecName, LPVOID lpData, short nDataLen)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szRecName</i>	LPSTR Pointer to the record name.
<i>lpData</i>	LPVOID Pointer to the storage location for the currency information.
<i>nDataLen</i>	short Length of storage location <i>lpData</i> .

Description

The XDbRecordUpdCurrency function updates the currency for a specific record type. For more on currency, see the Database Currency section in this manual.

See Also

DbRecordUpdCurrency

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, or E_NORECNAME. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;
    /* Note: 'currency_index is defined in DBMGR.H */
    struct currency_index currency;

    /* Get the currency for the client record */
    if( dwStatus = XDbRecordGetCurrency( hDb, "client", &currency, sizeof( struct
        currency_index)))
    {
        /* Currency not retrieved */
    }

    /* Other processing goes here... */

    /* Restore the currency for the client record */
    if( dwStatus = XDbRecordUpdCurrency( hDb, "client", &currency, sizeof( struct
        currency_index)))
    {
        /* Currency not updated */
    }
}
```

XDbSetGetFirst

Summary

DWORD FAR PASCAL XDbSetGetFirst(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

lpTarget **LPVOID** Pointer to the storage location for the retrieved member record.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbSetGetFirst function retrieves the first member of an owner/member set relation. The owner record must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

DbSetGetFirst

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;
    struct address saddress;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Get the first address record */
    if( dwStatus = XDbSetGetFirst( hDb, "client", "address", &saddress, sizeof( struct address)))
    {
        /* First member not found */
    }
}
```

XDbSetGetLast

Summary

DWORD FAR PASCAL XDbSetGetLast(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

lpTarget **LPVOID** Pointer to the storage location for the retrieved member record.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbSetGetLast function retrieves the last member of an owner/member set relation. The owner record must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

DbSetGetLast

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOTFOUND. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;
    struct address saddress;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Get the last address record */
    if( dwStatus = XDbSetGetLast( hDb, "client", "address", &saddress, sizeof( struct address)))
    {
        /* Last member not found */
    }
}
```

XDbSetGetNext

Summary

DWORD FAR PASCAL XDbSetGetNext(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, LPVOID lpTarget, short nTargetLen)

Parameters

<i>hDb</i>	HANDLE Identifies the database.
<i>szOwnerName</i>	LPSTR Pointer to the owner record name.
<i>szMemberName</i>	LPSTR Pointer to the member record name.
<i>lpTarget</i>	LPVOID Pointer to the storage location for the retrieved member record.
<i>nTargetLen</i>	short Length of storage location <i>lpTarget</i> .

Description

The XDbSetGetNext function retrieves the next member of an owner/member set relation. Both owner and member records must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

DbSetGetNext

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NONEXT. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;
    struct address address;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Get the first address record */
    if( dwStatus = XDbSetGetFirst( hDb, "client", "address", &address, sizeof( struct address)))
    {
        /* First member not found */
    }

    /* Get the next address record */
    if( dwStatus = XDbSetGetNext( hDb, "client", "address", &address, sizeof( struct address)))
    {
        /* Next member not found */
    }
}
```

XDbSetGetOwner

Summary

DWORD FAR PASCAL XDbSetGetOwner(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

lpTarget **LPVOID** Pointer to the storage location for the retrieved owner record.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbSetGetOwner function retrieves the owner record of a member record set relation. The member record must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

DbSetGetOwner

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOOWNER. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    DWORD dwStatus;
    struct client sclient;

    /* Set the currency to the first invoice record */
    if( dwStatus = DbRecordFindFirst( hDb, "invoice", "IInvoiceNbr"))
    {
        /* Invoice not found */
    }

    /* Get the client record for this invoice */
    if( dwStatus = XDbSetGetOwner( hDb, "client", "invoice", &sclient, sizeof( struct client)))
    {
        /* Client record not found */
    }
}
```

XDbSetGetPrev

Summary

DWORD FAR PASCAL XDbSetGetPrev(HANDLE hDb, LPSTR szOwnerName, LPSTR szMemberName, LPVOID lpTarget, short nTargetLen)

Parameters

hDb **HANDLE** Identifies the database.

szOwnerName **LPSTR** Pointer to the owner record name.

szMemberName **LPSTR** Pointer to the member record name.

lpTarget **LPVOID** Pointer to the storage location for the retrieved member record.

nTargetLen **short** Length of storage location *lpTarget*.

Description

The XDbSetGetPrev function retrieves the previous member of an owner/member set relation. Both owner and member records must have currency before this call. For more on currency, see the Database Currency section in this manual.

See Also

DbSetGetPrev

Return Value

A 0L is returned if no error occurred. Otherwise the return code can be E_WINALLOC, E_DOS, E_NORECNAME, E_INVALIDSET, E_NOCURRENT, or E_NOPREV. See the 'Error Messages' section for more detail on these values.

Example

```
#include <windows.h>
#include "dbmgr.h"

void Function( HANDLE hDb)
{
    LONG IClientNbr = 1000L;
    DWORD dwStatus;
    struct address address;

    /* Set the currency to client record #1000 */
    if( dwStatus = XDbRecordFindByKey( hDb, "client", "IClientNbr", &IClientNbr, sizeof( LONG)))
    {
        /* Record not found */
    }

    /* Get the last address record */
    if( dwStatus = XDbSetGetLast( hDb, "client", "address", &saddress, sizeof( struct address)))
    {
        /* Last member not found */
    }

    /* Get the previous address record */
    if( dwStatus = XDbSetGetPrev( hDb, "client", "address", &saddress, sizeof( struct address)))
    {
        /* Previous member not found */
    }
}
```

Error Messages

This section describes error messages that you may encounter when developing a program using CDB For Windows.

CDB Run-Time Error Messages

The CDB For Windows C-API function calls return a DWORD value indicating the success or failure of a particular database call. A 0L is returned if the function was a success. A non-zero value is returned if an error occurred.

If the return value is non-zero, the LOWORD contains the error code and the HIWORD contains the extended error code if one exists. Use the LOWORD and HIWORD macros to decipher the return value as indicated in the example below:

Example

```
void Function(void)
{
    HANDLE hDb;

    dwStatus = DbOpen( hParentWnd, ".\\", "sampledb.dbd", &hDb);
    if( dwStatus)
    {
        nErrorCode = LOWORD( dwStatus);
        nExtErrorCode = HIWORD( dwStatus);

        // Decipher error code
        .
        .
        .
    }
}
```

The following list displays the possible error codes that can be returned and a brief explanation of each.

<u>Error Code</u>	<u>Description</u>
E_TESTDRIVE	You are using a 'Test Drive' version of CDB For Windows. The 'Test Drive' version limits the number of records that can be added to a database to 50.
E_WINALLOC	A Windows GlobalAlloc or GlobalLock error has occurred. Most likely you are out of global memory.

<u>Error Code</u>	<u>Description</u>
E_LOADMODULE	A Windows LoadModule error has occurred. The HIWORD of the return value contains the specific LoadModule error code. See the Windows SDK documentation for more information about the LoadModule return codes.
E_INVALIDCASE	Contact Daytris technical support. An internal switch statement does not contain a valid case. You should never see this error code.
E_DOS	An MS-DOS error has occurred. The HIWORD of the return value contains the specific DOS error number. It is the global 'errno' value. See ERRNO.H and/or the Microsoft C documentation for more information.
E_NORECNAME	The record name passed, <i>szRecName</i> , is not a valid record type.
E_NOFLDNAME	The field name passed, <i>szFldName</i> , is not a valid field for the record.
E_INVALIDSET	The owner and member names passed to the function do not have a set relationship between the two. To create a set relationship between two records, use the CONNECT keyword in the DDL file.
E_NOTAKEY	The field name passed to the function is not a key field in the record. Use the KEY keyword in the DDL file to define key fields.
E_NOTFOUND	The record was not found.
E_NEXTGUESS	The record was not found, but the next closest match to the key value passed was found. If a DbRecordFindByKey or XDbRecordFindByKey call was made, currency is set to this 'next guess' record. If a DbRecordGetByKey or XDbRecordGetByKey call was made, the 'next guess' record is returned.
E_NOCURRENT	There is no current record for the record name specified. e.g. This error value will be returned if a DbRecordGetNext call is made before the record requested has currency. One way to set currency in this case would be to make a DbRecordFindFirst call. There are a number of other cases where this error code could be returned.

<u>Error Code</u>	<u>Description</u>
E_NONEXT	The next record was not found. e.g. DbRecordGetNext(...).
E_NOPREV	The previous record was not found. e.g. DbRecordGetPrevious(...).
E_NOOWNER	This error can only occur with a DbSetGetOwner call. If an owner record is not found, this value is returned.
E_ALREADYOPEN	This database has already been opened by another application. A database cannot be shared by different applications. Future versions of CDB For Windows will support mult-application access to the same database.

DDL Error Messages

The following lists contain a description of error and warning messages that may be encountered during the execution of the Data Definition Language Parse utility (DDL.P.EXE):

<u>Error Number</u>	<u>Description</u>
100	Unexpected end of file reached.
101	Unexpected token. DDL breaks the DDL file into tokens. A token can be a bracket, keyword, semicolon, variable, constant, etc. It combines the tokens and matches them against predefined patterns. If a pattern has no match, this error is returned.
102	Expecting semicolon.
103	Expecting "struct" keyword. DDL is expecting a structure definition to begin.
104	Expecting identifier. An identifier can be a structure name or field name.
105	Expecting '{'. Expecting a left brace.
106	Constant too big. A constant is a number. The maximum constant size allowed is 10 digits.
107	Structure already defined.
108	Invalid constant. The interpreted value of the constant is zero.
109	Maximum size of a constant is 65535.
110	Maximum size of a field is 65535.
111	Maximum size of a record is 65535.
112	Connection already made to

'record name'. You cannot to the same record more than once.

<u>Error Number</u>	<u>Description</u>
113	Record does not exist. This error will occur when you try to CONNECT to a record that is not defined within the DDL file.
114	Cannot connect structure to itself.
115	Connect key field not found. If you are ordering the sets using the 'CONNECT record_name KEY key_field_name' convention, the key_field_name is not found within the record_name structure definition in the DDL file.

<u>Warning Number</u>	<u>Description</u>
100	"prefix" not defined, assuming "test". The PREFIX keyword was not used to define the prefix used to derive database file names. A "test" prefix is used as a default.
101	Prefix too long, truncating to 'identifier'. The maximum length of a PREFIX is 5 characters.
102	Identifier too long. The maximum length of an identifier is 31 characters. Extra characters will be truncated.