

CODBCRecordset class

CODBCRecordset class is intended to be full replacement of all ClassWizard generated CRecordset derived classes in MFC projects.

This article was contributed by **Stefan Chekanov**.

Platform: VC 5.0 & VC 6.0

- [Introduction](#)
- [Examples of how to use CODBCRecordset](#)
- [How does CODBCRecordset work?](#)

Introduction

In a usual MFC database project we have a lot of CRecordset derived classes generated by ClassWizard. The presented here CODBCRecordset class is a very easy to use replacement of all these CRecordset classes. CODBCRecordset is derived from CRecordset but it does not have hardcoded inside the number and the type of the database fields.

I have seen some implementations trying to solve this problem. But CODBCRecordset has at least two advantages:

1. CODBCRecordset can be used not only to get values from the database, but to write values into the database too using the MFC way.
2. The other implementations cannot open simultaneously more than one recordset through one database connection when the database is MS SQL Server. In some cases this could be very big problem because opening a new connection is time and resources expensive.

Because CODBCRecordset is derived from CRecordset and uses its data exchange mechanism it is fully compatible with MFC ODBC class CDatabase.

Each field value is stored in a CDBField object. CDBField class is inherited from CDBVariant and has added some extra functionality to be easy to use.

CODBCRecordset and CDBField classes support all database data types. CDBField makes implicit type conversion where is appropriate to supply the data in the requested format.

Here is a list of CODBCRecordset and CDBField methods:

CODBCRecordset class	
Constructor	The same as for CRecordset accepting CDatabase*
BOOL Open(LPCTSTR lpszSQL, UINT nOpenType, DWORD dwOptions);	Open the recordset lpszSQL is a SQL statement that returns recordset e.g. SELECT * FROM tablename nOpenType is CRecordset open type, see CRecordset::Open() dwOptions is CRecordset options, see CRecordset::Open() Note that lpszSQL and nOpenType have exchanged their positions compared to CRecordset::Open()
short GetODBCFieldCount()	Returns number of the fields (columns) in the recordset. This method is defined in CRecordset.
int GetFieldID(LPCTSTR)	Returns a field index by given field name. It is case insensitive. CRecordset::GetFieldIndexByName() works, but is case sensitive.
CString GetFieldName(int)	Returns a field name by given field index

<p>CDBField& Field(LPCTSTR) CDBField& Field(int)</p>	<p>Through this method you can get a reference to the internal CDBField object corresponding to the specified column (See the CDBField class table for details about CDBField).</p> <p>There are two forms of the method - with argument of type LPCTSTR szName - specifying the name of the column, and int nID - specifying the index of the coulmn in the recordset.</p> <p>These methods can be used as lvalue in expressions. Thus you can write values to the database.</p>
<p>CDBField& operator(LPCTSTR) CDBField& operator(int)</p>	<p>The function operator is defined for easy of use of the class and just calls the corresponding Field() method</p> <p>There are two forms of the function operator - with argument of type LPCTSTR szName - specifying the name of the column, and int nID - specifying the index of the coulmn in the recordset.</p> <p>These can be used as lvalue in expressions. Thus you can write values to the database.</p>
<p>bool GetBool() unsigned char GetChar() short GetShort() int GetInt() long GetLong() float Getfloat() double GetDouble() COleDateTime GetDate() CString GetString() CLongBinary* GetBinary()</p>	<p>All of these methods do the appropriate type conversions depending on their return value and the type of the underlying data in the database.</p> <p>These methods just call Field().AsXXX() (See the CDBField class table for details about CDBField).</p> <p>There are two forms of these methods - with argument of type LPCTSTR szName - specifying the name of the column, and int nID - specifying the index of the coulmn in the recordset.</p> <p>These cannot be used as lvalue in expressions.</p>

CDBField class	
Constructors	No public constructors. CDBField cannot be instantiated except by CODBCRecordset to be used in internal structures to support data exchange. These objects are accessible through CODBCRecordset methods.
<p>bool AsBool() unsigned char AsChar() short AsShort() int AsInt() long AsLong() float AsFloat() double AsDoble() COleDateTime AsDate() CString AsString() CLongBinary* AsBinary()</p>	<p>All of these methods do the appropriate type conversions depending on their return value and the type of the underlying data in the database (See the AsXXX methods table for data conversion rules).</p> <p>There is no data type <i>Int</i> but AsInt() is equal to AsLong()</p>
assignment operators	There are defined assignment operators accepting bool, unsigned char, short, int, long, COleDateTime and CString . So CDBField objects can be lvalues. There is no assignemt operator accepting CLongBinary because MFC does not support writing CLongBinary values into database. These assignment operators do appropriate conversions to the underlying database column data type except CLongBinary (See the assignment operators table for data conversion rules).
const CString& GetName()	Returns the field name this object corresponds to.
<p>bool IsNull() bool IsBool() bool IsChar() bool IsShort() bool IsInt() bool IsLong() bool IsFloat() bool IsDouble() bool IsNumber() bool IsDate() bool IsString() bool IsBinary()</p>	<p>Each of these return true if the field contains a value of the corresponding data type.</p> <p>There is no data type <i>Number</i> but IsNumber() returns true if IsShort() IsLong() IsFloat() IsDouble().</p> <p>There is no data type <i>Int</i> but IsInt() returns true if IsLong() returns true.</p>

Conversions made by AsXXX methods										
	Values in the database									
	NULL	BOOL	UCHAR	SHORT	LONG	SINGLE	DOUBLE	DATE	STRING	BINARY
AsBool	false	*	*	*	*	*	*		*	
AsChar	0x32	*	*	*	*	*	*		*	
AsShort	0	*	*	*	*	*	*		*	
AsInt	0	*	*	*	*	*	*		*	
AsLong	0	*	*	*	*	*	*		*	
AsFloat	0.0	*	*	*	*	*	*		*	
AsDouble	0.0	*	*	*	*	*	*		*	
AsDate	null	invalid	invalid	*	*	*	*	*	*	
AsString	empty	*	*	*	*	*	*	*	*	*
AsLongBinary	NULL									*

Empty cells indicate the conversion **is not** available, thus code asserts.

Cells marked with * indicate conversion **is** available (See the [Conversion algorithms](#) table for data conversion rules).

Conversions made by assignment operators									
Database Field Type	Argument of the assignment operator								
	bool	unsigned char	short	int	long	float	double	COleDateTime	String
NULL									
BOOL	*	*	*	*	*	*	*		*
UCHAR	*	*	*	*	*	*	*		*
SHORT	*	*	*	*	*	*	*		*
LONG	*	*	*	*	*	*	*		*
SINGLE	*	*	*	*	*	*	*		*
DOUBLE	*	*	*	*	*	*	*		*
DATE								*	*
STRING	*	*	*	*	*	*	*	*	*
BINARY									

Empty cells indicate the conversion **is not** available, thus code asserts.

Cells marked with * indicate conversion **is** available (See the [Conversion algorithms](#) table for data conversion rules).

Conversion algorithms	
String to Bool	comparing the first character of the string with 'T'
Char to Bool	comparing the character with 'T'
Bool to String	String = (bVal) ? 'T' : 'F'
Bool to Char	Char = (bVal) ? 'T' : 'F'
String to Number	appropriate atoX() function
Number to String	CString::Format() method using the appropriate format specifier string
String to Date	COleDateTime::ParseDateTime() method
Date to String	COleDateTime::Format() method

Examples of how to use CODBCRecordset

You should include the files `ODBCRecordset.h` and `ODBCRecordset.cpp` in your project.

I usually include this line in my `StdAfx.h` file.

```
#include "ODBCRecordset.h"
```

Here is a simple code showing how CODBCRecordset can be used.

```
//////////////////////////////////////
CDatabase      db;
//      This connect string will pop up the ODBC connect dialog
CString       cConnect = "ODBC;";
db.Open( NULL, //      DSN
        FALSE, //      Exclusive
        FALSE, //      ReadOnly
        cConnect, //      ODBC Connect string
        TRUE //      Use cursor lib
    );

COleDateTime  dOrderDate;

CODBCRecordset rs( &db );
rs.Open( "SELECT * FROM Orders \
        WHERE ORDER_DATE > 'jan 1 2000' \
        ORDER BY ORDER_DATE" );
for( ; ! rs.IsEOF(); rs.MoveNext() )
{
//      The choice is yours. You may choose whatever way
//      you want to get the values
//
//      These return COleDateTime value
dOrderDate = rs.GetDate( "ORDER_DATE" );
dOrderDate = rs.Field("ORDER_DATE").AsDate();

//      These make implicit call to AsDate()
dOrderDate = rs("ORDER_DATE");
dOrderDate = rs.Field("ORDER_DATE");

//      Now edit the fields in the recordset
rs.Edit();
rs("ORDER_DATE") = "jan 1 1999"; // Implicit conversion
rs.Field("ORDER_DATE") = "jan 1 1999"; // Implicit conversion
rs.Update();
} //      for(...)
//////////////////////////////////////
```

If **ORDER_DATE** is stored in the database as **datetime** or compatible data type the value will be get directly.

If **ORDER_DATE** is stored in the database as **string** or compatible data type (char, varchar) the value will be converted via `COleDateTime::ParseDateTime()` method. If conversion fails, **dOrderDate** will be set to **COleDateTime::invalid**.

When opening a resultset generated by join statements it is possible to get 2 or more columns that have the same name. CODBCRecordset leaves the name of the first column intact but other repeated columns are renamed with adding the number this columns repeats the name. Not repeated column names are left intact. E.g.

```
SELECT * FROM Orders, Customers WHERE Orders.CUST_ID = Customers.ID
```

If the table *Orders* have a column with name *ID* and *Customers* have a column with name *ID*, CODBCRecordset will rename *ID* from Customers to **ID2** and all other not repeating column names will be intact.

Well, here is a tip: Rename columns manually to be sure what name they have, e.g.

```
SELECT Orders.*, Customers.ID as CUSTOMERS_ID
FROM Orders, Customers
WHERE Orders.CUST_ID = Customers.ID
```

How does CODBCRecordset work?

CODBCRecordset allocates storage for all fields in the resultset and uses MFC *Record Field eXchange* mechanism like it has been inherited from CRecordset using ClassWizard.

That's all.