

ADO – LISP Library (ADOLISP)

Revision 2.52
April, 2007

Jon Fleming
The Fleming Group
jonf@fleming-group.com

**Important: Users of ADOLISP 2.4 and previous read section
4.1.2.2.2!**

See also Vista Compatibility in section 3.1.3.

Table of Contents

1. INTRODUCTION	1
1.1 COPYRIGHT AND DISCLAIMER	1
1.2 DBL FUNCTIONS THAT DO NOT APPEAR IN ADOLISP	1
1.3 OTHER CHANGES COMPARED TO DBL	2
1.4 WHAT THIS LIBRARY IS NOT SUPPOSED TO BE	2
2. INSTALLATION.....	3
2.1 REQUIREMENTS	3
2.2 PROCEDURE	3
3. USING ADOLISP	4
3.1 IMPORTANT NOTES	4
3.1.1 Using Double Quotes around Table or Column Names in SQL Statements.....	4
3.1.2 When an SQL or Database Error Occurs.....	4
3.1.3 Vista Compatibility	4
3.2 LOADING ADOLISP	5
4. ADOLISP FUNCTIONS.....	6
4.1 CORE FUNCTIONS.....	6
4.1.1 ADOLISP_ConnectToDB - Connect to a Database	6
4.1.1.1 Usage	6
4.1.1.2 Return Value	6
4.1.1.3 OLEDB Connect Strings.....	6
4.1.1.3.1 General Connect Strings using ODBC	7
4.1.1.3.2 Connect strings for Access	7
4.1.1.3.3 Connect Strings for SQL Server.....	7
4.1.1.3.4 Connect Strings for FoxPro.....	8
4.1.1.3.5 Connect Strings for Excel	8
4.1.1.3.6 Further Information	8
4.1.2 ADOLISP_DoSQL - Execute One SQL Statement.....	9
4.1.2.1 Usage	9
4.1.2.2 Return Value	9
4.1.2.2.1 When the function fails	9
4.1.2.2.2 When the SQL statement is a “non-cursor” statement	9
4.1.2.2.3 When the SQL statement is a “cursor” statement	9
4.1.3 ADOLISP_DisconnectFromDB - Disconnect From a Database.....	10
4.1.3.1 Usage	10
4.1.3.2 Return Value	10
4.1.3.3 Important Note	10
4.2 UTILITY FUNCTIONS.....	10
4.2.1 ADOLISP_ErrorPrinter – Print the Last SQL Statement and Errors	10
4.2.1.1 Usage	10
4.2.1.2 Return Value	10
4.3 NAME-HANDLING AND INFORMATION FUNCTIONS	11
4.3.1 ADOLISP_GetTablesAndViews – Get all the table and view names.....	11
4.3.1.1 Usage	11
4.3.1.2 Return Value	11
4.3.2 ADOLISP_GetColumns – Get information on the columns in a table.....	11
4.3.2.1 Usage	11
4.3.2.2 Return Value	11
5. ADOLISP GLOBAL VARIABLES	12
5.1 ADOLISP_DoNotForceJetODBCPARSING – JET DRIVER PROPERTY	12
5.2 ADOLISP_FIELDSPROPERTIESLIST – PROPERTIES OF RETRIEVED FIELDS	13
5.2.1 Changes in Revision 2.31.....	14

6.	EXAMPLES	14
7.	CONNECTING TO EXCEL	14
7.1	SETTING UP AN EXCEL SPREADSHEET	15
7.2	EXCEL LIMITATIONS	15
7.3	FURTHER INFORMATION	16
8.	HINTS AND TIPS	16
8.1	DATABASE TABLE AND COLUMN NAMES	16
8.2	AUTONUMBER FIELDS	16
8.3	SETTING DATE AND TIME FIELDS	16
APPENDIX A	SQL/92 RESERVED WORDS	18
APPENDIX B	VALUES AND MEANINGS OF THE FIELD TYPE	20
APPENDIX C	VALUES AND MEANINGS OF THE FIELD ATTRIBUTES	21

1. Introduction

I have been involved with connections between AutoCAD and databases for several years. The AutoCAD SQL Interface (ASI) scheme for connecting AutoCAD to databases in Release 14 and previous was tricky, fragile, and error-prone. I wrote a free library (DBL, still available on the download page at <http://www.fleming-group.com>) to make it easier to connect to a database in Release 14 and previous.

When AutoCAD 2000 was released, Autodesk added the capability of using ActiveX Data Objects (ADO) for database connections, and announced that ASI would probably be dropped in a future release. I updated DBL to work with AutoCAD 2000. However, DBL is large and complex (since it supports AutoCAD 12 through AutoCAD 2000). The ADO paradigm is very different. Updating DBL to use either ASI or ADO is impractical. Therefore, I will no longer update DBL, and am releasing a similar library (ADOLISP) for AutoCAD 2000 and higher that uses only ADO.

ADO is much simpler than ASI, but there are some intricacies involved in using ADO from LISP. The major reason for this library is to spare others the pain of learning those intricacies.

The interface to ADOLISP is very similar to the interface to DBL, and it should be easy to convert programs that use DBL into programs that use ADOLISP. I strongly recommend doing this conversion unless your program must support Release 14 or earlier. In the (hopefully rare) case that your program must support Release 14 or earlier *and* a version of AutoCAD that does not support ASI, you will just have to use both libraries and determine which one to call in your program.

1.1 Copyright and Disclaimer

The code and documentation are Copyright © 1999-2003 by The Fleming Group

Permission to use, copy, modify, and distribute this software for any purpose and without fee is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and the limited warranty and restricted rights notice below appear in all supporting documentation.

If problems are encountered, I will do what I can to correct them. However, **The Fleming Group provides this program "as is" and with all faults. The Fleming Group specifically disclaims any implied warranty of merchantability or fitness for a particular use. The Fleming Group does not warrant that the operation of the program will be uninterrupted or error free.**

1.2 DBL functions that do not appear in ADOLISP

The following functions from DBL are not present in ADOLISP and there is no equivalent function in ADOLISP:

- DBLGetEnvironmentNames
- DBLLookupTableName

- DBLLookupColumnName
- DBLGetColumnNames

The ADOLISP_GetTablesAndViews function provides some of the functionality of DBLGetEnvironmentNames.

1.3 Other changes compared to DBL

The first argument of DBLConnectToDB is no longer used; it was only used in R12.

The second argument of DBLConnectToDB is equivalent to the first argument of ADOLISP_ConnectToDB, but the meaning is different (as discussed in the section on ADOLISP_ConnectToDB).

ADOLISP_ConnectToDB does not print any errors or any alert boxes. If an error occurs, the calling program must call (ADOLISP_ErrorPrinter) at an appropriate time or otherwise obtain the value of the ADOLISP_ErrorList variable.

ADOLISP_DoSQL does not use one of the arguments that was required by DBLDoSQL. There is no argument for whether or not to print errors. ADOLISP_DoSQL never prints errors; the calling routine must call (ADOLISP_ErrorPrinter) at an appropriate time or otherwise obtain the value of the ADOLISP_ErrorList variable.

1.4 What this Library is Not Supposed to Be

This is not intended to teach AutoLISP programming (although I hope I have provided a few examples of good programming practice). To keep the example routine simple, I have not included an error handler or any dialog box interfaces.

This is also not an SQL tutorial. To interact with the database you must write SQL statements. There are some examples of common SQL statements in the examples, and this may be enough to get you started.

To learn SQL with no previous experience, "SQL Clearly Explained", Jan L. Harrington, API Professional, ISBN 0-12-326426-X is a good starting point. If you know some SQL, "A Guide to the SQL Standard, Fourth Edition", C. J. Date with Hugh Darwin, Addison-Wesley, ISBN 0-201-96426-0 is a good reference.

You can also learn some SQL from Microsoft Access by opening an existing query in "design view" and clicking on the "SQL" button. You should be aware that some of the SQL statements that Access will generate are more complex than is required (this is a common attribute of machine-generated programming statements). You should also be aware that Microsoft Access SQL differs from standard SQL in a few ways:

- Where Access uses double quotes (") around strings, standard SQL uses single quotes (').
- Where Access uses square brackets ([]) around identifier names with spaces or identifier names that are SQL reserved words, standard SQL uses double quotes (").

- Where Access uses * inside strings as a multiple-character wild-card character, standard SQL uses %. (Standard SQL uses * as a wild card outside of strings).
- Where Access uses ? inside strings as a single-character wild-card character, standard SQL uses _ (underscore).

So, a query to select all rows in “My Table” in which the entry in the “Section” column starts with F followed by any one character followed by T followed by any string would appear in Access as:

```
select * from [My Table] where [Section] like "F?T*"
```

(note that “section” is a reserved word in SQL/92)

and would be, in standard SQL:

```
select * from "My Table" where "Section" like 'F_T%'
```

Some versions of the Microsoft Jet database engine accept only square brackets around delimited identifiers by default. ADOLISP changes this behavior, automatically setting the Jet database engine to accept standard SQL when the Jet database engine is being used. This allows writing SQL statements in your program without worrying about which engine is used to connect to the database. However, changing this behavior sometimes causes problems, so you can tell ADOLISP not to change the behavior (see “ADOLISP_DoNotForceJetODBCParsing – Jet Driver Property” on page 12). You should first try using standard SQL syntax without setting ADOLISP_DoNotForceJetODBCParsing; if that fails, try standard SQL syntax but set ADOLISP_DoNotForceJetODBCParsing to something non-nil; finally, if that fails, try Access syntax and set ADOLISP_DoNotForceJetODBCParsing to non-nil.

2. Installation

2.1 Requirements

⇒ AutoCAD 2000 or higher.

2.2 Procedure

ADOLISP is totally contained in ADOLISP.LSP. All you must do is copy ADOLISP.LSP to a convenient location.

Before using ADOLISP, you must write AutoLISP code to call the functions (ADOLISP_Example.LSP contains several examples) and set up the connection from AutoCAD to the database. You may either set up a UDL file using the AutoCAD “dbConnect” command or you can use an OLEDB Connection String .

OLEDB connection strings are discussed in the section on using ADOLISP_ConnectToDB.

3. Using ADOLISP

3.1 Important Notes

3.1.1 Using Double Quotes around Table or Column Names in SQL Statements

The SQL standard requires double quotes around table or column names in two circumstances:

- When the table or column name is an SQL Reserved Word (see “Appendix A SQL/92 Reserved Words” on page 18)
- When the table or column name is not a “Regular Identifier”. A Regular Identifier contains only letters, digits, and underscores. For example, a column name that contains a space is not a Regular Identifier.

SQL allows double quotes around table or column names where double quotes are not required.

In ASI, more stringent requirements were imposed, and it was complicated to meet those requirements. ADO appears to adhere to the SQL standard; you only need double quotes as noted above.

3.1.2 When an SQL or Database Error Occurs

When an error occurs in an ADOLISP function, as much information as possible is stored in the global variable `ADOLISP_ErrorList`. The error is never printed immediately to the screen, in case a dialog box is displayed (which would cause the printing to fail). Your program *must* check the return values from ADOLISP functions and, if an error has occurred, call `(ADOLISP_ErrorPrinter)` when it is appropriate to print the errors or otherwise obtain the value of the `ADOLISP_ErrorList` variable.

`ADOLISP_DoSQL` stores the text of the last SQL statement in the global variable `ADOLISP_LastSQLStatement`. It is often useful to inspect the value of this variable when problems occur.

3.1.3 Vista Compatibility

ADOLISP versions 2.52 and up are compatible with AutoCAD 2008 running in Vista. Previous versions are not compatible with AutoCAD 2008 running in Vista and are probably not compatible with other AutoCAD versions in Vista. Other versions of AutoCAD have not been tested in Vista, but if you can get some other version running ADOLISP will probably work.

Note that it is not possible to use ADOLISP to access a database stored in the root directory of the system drive, due to Vista’s new security features. Accessing databases stored in the root directory of other drives has not been tested.

3.2 Loading ADOLISP

ADOLISP.LSP does not have to be stored in a directory that AutoCAD searches for files. However, if you do not put it in a directory that AutoCAD searches for files, you will have to load ADOLISP.LSP manually before using it.

There are many techniques for loading LISP support routines such as ADOLISP. Here's a method I like:

```
;; If ADOLISP.LSP is already loaded or we can find it
;; and load it ...
(if (or ADOLISP_ConnectToDB
      (and (findfile "ADOLISP.LSP") (load "ADOLISP.LSP")))
    ) ;_ end or
    ;; Then we can go ahead and define all our functions. See the
    ;; end of the file for what happens if we don't have
    ;; ADOLISP.lsp
    (progn
      (defun C:FunctionOne ()
        .
        .
        .
      )
      (defun C:FunctionTwo ()
        .
        .
        .
      )
      .
      .
      .
    )
    ;; If we got here we can't find ADOLISP.LSP. If we are being
    ;; automatically loaded by the Autodesk (autoload ...)
    ;; function, one of our (c:...) functions is about to
    ;; be called from a function of the same name, so we have
    ;; to define all our (c:...) functions with dummy
    ;; bodies
    (progn
      (defun C:FunctionOne ()
        (alert
          "Can't find ADOLISP.LSP!\nFix the problem and reload."
        )
      )
      (defun C:FunctionTwo ()
        (alert
          "Can't find ADOLISP.LSP!\nFix the problem and reload."
        )
      )
      .
      .
      .
    )
  )
```


(println)

4. ADOLISP Functions

4.1 Core Functions

4.1.1 ADOLISP_ConnectToDB - Connect to a Database

4.1.1.1 Usage

(setq ConnectionObject (ADOLISP_ConnectToDB ConnectString UserName Password))

ConnectString	A string containing <i>either</i> an OLEDB Connection String (discussed below) <i>or</i> the name of a UDL file (including the “.udl” extension) that is in the current directory, on the search path, in the Data Source Location (usually the “Data Links” subdirectory of the main AutoCAD directory), or includes a fully qualified path.
UserName	A string containing the user name to use when logging on to the database. This is often “admin”. Depending on the value of the first argument, this argument might not actually be used, but it must be present.
Password	A string containing the password to use when logging on to the database. This is often a null string. Depending on the value of the first argument, this argument might not actually be used, but it must be present.

4.1.1.2 Return Value

If it fails, NIL. If this happens, call (ADOLISP_ErrorPrinter) to print the error description(s) or otherwise obtain the value of the ADOLISP_ErrorList variable.

If it succeeds, a “Connection Object”. ***You must save the return value of ADOLISP_ConnectToDB*** so you can pass it to the other functions.

4.1.1.3 OLEDB Connect Strings

It is often convenient to use an OLEDB Connect String, because then your program depends only on having ADOLISP and does not require any other files or setup to connect to the database. I don’t know of a good source for learning about connect strings in general, but I will provide what information I have.

One key item is that you should *never* put a space in an OLEDB Connect String unless it is required.

The first item in an OLEDB Connect String is almost always the “provider name”. The default provider is the ODBC system. However, I like to always put in the provider name which, for ODBC, is MSDASQL.

I list several different connection strings in the following sections. There is a huge compendium of connection strings at http://www.able-consulting.com/ADO_Conn.htm.

4.1.1.3.1 General Connect Strings using ODBC

If you are connecting to a database using ODBC, you can use a Data Source Name (DSN) (which must be set up in the ODBC Connection Manager before using it) or you can do a “DSN-less” connection (which requires no previous setup in the ODBC Connection Manager). I like DSN-less connections myself, but here is an example of using a DSN named “myconnection” that has been set up in the ODBC manager:

```
"Provider=MSDASQL;DSN=myconnection"
```

(Note: if you are connecting to Excel, the default is to open the file read-only. When setting up the ODBC connection, pick the “Options” button and un-check the “Read Only” check box).

4.1.1.3.2 Connect strings for Access

To set up a DSN-less connection to a Microsoft Access database through older versions of ODBC:

```
"Provider=MSDASQL;Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\ADOLISP_test.mdb"
```

Or using more recent versions of ODBC:

```
Provider=MSDASQL.1;Persist Security Info=False;Extended Properties="Driver={Microsoft Access Driver (*.mdb)};DBQ=C:\ADOLISP_test.mdb"
```

Note that the “Driver” name is the name that appears in the ODBC manager in the “Driver” column. Note also that, when you put this (or any similar) string into an AutoLISP program as a literal string, you must change “\” to “\\”.

To set up a non-ODBC connection to a Microsoft Access database using the Jet 4.0 engine (which is a little bit faster than ODBC):

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\ADOLISP_test.mdb;Persist Security Info=False"
```

(the “Persist Security Info” setting is required when using ADO from AutoCAD).

Or the same thing using the Jet 3.51 engine

```
"Provider=Microsoft.Jet.OLEDB.3.51;Data Source=C:\ADOLISP_test.mdb"
```

4.1.1.3.3 Connect Strings for SQL Server

For SQL Server using ODBC (I haven’t tested these):

```
"Provider=MSDASQL;Driver={SQL
Server};Server=servername;Database=dbname;UID=userid;PWD=password"
```

```
Provider=MSDASQL.1;Persist Security Info=False;Extended
Properties=";Driver={SQL
Server};Server=servername;Database=dbname;UID=userid;PWD=password"
```

(replacing servername, dbname, userid, and password with appropriate values).

Or, for SQL Server without ODBC:

```
"Provider=SQLOLEDB;Data Source=servername;Initial
Catalog=dbname;User ID=userid;Password=password"
```

4.1.1.3.4 Connect Strings for FoxPro

For Visual FoxPro (I haven't tested this one either):

```
"Provider=vfpoledb.1;Data Source=TasTrade.dbc"
```

4.1.1.3.5 Connect Strings for Excel

(See Connecting to Excel on page 14 for information on the proper setup in Excel).

For connecting to Excel through ODBC:

```
"Driver=Microsoft Excel Driver
(*.xls);DBQ=D:\test.xls;ReadOnly=False"
```

For connecting to Microsoft Excel through Jet 4.0:

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\test.xls;Extended properties=Excel 8.0;Persist
Security Info=False"
```

(For Excel 95, replace "Excel 8.0" with "Excel 5.0".)

You cannot connect to Excel using Jet 3.51.

If you have difficulties connecting to Excel, try using the ADOLISP_DoNotForceJetODBCParsing variable as described in "ADOLISP_DoNotForceJetODBCParsing – Jet Driver Property" on page 12.

4.1.1.3.6 Further Information

You might be able to find more information at <http://msdn.microsoft.com/>.

4.1.2 ADOLISP_DoSQL - Execute One SQL Statement

4.1.2.1 Usage

(ADOLISP_DoSQL ConnectionObject SQLStatement)

ConnectionObject The return value of ADOLISP_ConnectToDB.

SQLStatement A string containing an SQL statement to execute.

Note that ADOLISP_DoSQL does not support “replaceable parameters”. The SQL statement passed to ADOLISP_DoSQL must be complete and ready to be executed.

Note: When using a “Where” condition that involves Microsoft Access yes/no fields, or when inserting a value into a yes/no field, use the (case-sensitive) strings “True” or “False”.

4.1.2.2 Return Value

4.1.2.2.1 When the function fails

ADOLISP_DoSQL returns NIL when it fails. . If this happens, call (ADOLISP_ErrorPrinter) to print the error description(s) or otherwise obtain the value of the ADOLISP_ErrorList variable.

4.1.2.2.2 When the SQL statement is a “non-cursor” statement

A “non-cursor” statement is one that can not result in any rows being extracted from the database and returned. A reasonable rule of thumb is that any statement that does *not* start with “select ...” is a non-cursor statement.

When ADOLISP_DoSQL successfully executes a non-cursor statement, it returns T (that is, non-nil). ***This is a change from previous versions, in which ADOLISP_DoSQL returned an integer number of rows affected unless it was running in AutoCAD 2000. This change was required for compatibility with Microsoft patch KB927779.***

4.1.2.2.3 When the SQL statement is a “cursor” statement

A “cursor” statement is one that can result in rows being extracted from the database and returned. A reasonable rule of thumb is that any statement that starts with “select ...” is a cursor statement.

When ADOLISP_DoSQL successfully executes a cursor statement, it returns a list containing the results. *This list will always contain at least one sub-list*; the minimum length will be 1. The first item in the returned value will be a list the names of the columns that were or could have been retrieved. You often will not care about the contents of this first sub-list

If at least one row was retrieved, the second item in the return value will be a list containing the fields in the same order that they were listed in the SQL statement (the same order as the column names in the first list) or, if no columns were explicitly named in the SQL statement, the order in which the columns are defined in the database. The first field is number 0. If more than one row

was retrieved, there will be further sub-lists in the return value, one for each retrieved row. The number of rows retrieved is:

(1- (length ADOLISP_DoSQLReturnValue))

Note: ADOLISP converts Microsoft Access yes/no field values into the strings “True” or “False” in the returned value list.

Note: ADOLISP converts dates and times into strings. Dates are returned in United States month/day/year format (the month and day are always two digits and the year is always four digits). Times are returned in 24-hour format, hours:minutes:seconds. All time values are always two digits.

4.1.3 ADOLISP_DisconnectFromDB - Disconnect From a Database

4.1.3.1 Usage

(ADOLISP_DisconnectFromDB ConnectionObject)

ConnectionObject The return value of ADOLISP_ConnectToDB.

4.1.3.2 Return Value

Always returns T.

4.1.3.3 Important Note

ADOLISP_DisconnectFromDB releases the connection object but *cannot* and *does not* set the connection object to nil. The value of the connection object will be non-nil after executing this function. If your code *ever* tests for a non-nil connection object, you should set the connection object to nil immediately after executing ADOLISP_DisconnectFromDB.

4.2 Utility Functions

4.2.1 ADOLISP_ErrorPrinter – Print the Last SQL Statement and Errors

4.2.1.1 Usage

(ADOLISP_ErrorPrinter)

ADOLISP_ErrorPrinter prints the most-recently-executed SQL statement (if there is one) to the AutoCAD text screen.

If execution of the most-recently-executed SQL statement produced an SQL error or errors, ADOLISP_ErrorPrinter prints the error message or messages to the AutoCAD text screen.

4.2.1.2 Return Value

None.

4.3 Name-Handling and Information Functions

4.3.1 ADOLISP_GetTablesAndViews – Get all the table and view names

4.3.1.1 Usage

(setq NameList (ADOLISP_GetTablesAndViews ConnectionObject))

ConnectionObject The return value of ADOLISP_ConnectToDB.

4.3.1.2 Return Value

If it fails, ADOLISP_GetTablesAndViews returns NIL. If this happens, call (ADOLISP_ErrorPrinter) to print the error description(s).

If it succeeds, ADOLISP_GetTablesAndViews returns a list of two lists of strings. The first list contains the names of all the tables in the database. The second list contains the names of all the views (called “queries” in Microsoft Access) in the database.

4.3.2 ADOLISP_GetColumns – Get information on the columns in a table

4.3.2.1 Usage

(setq ColumnList (ADOLISP_GetColumns ConnectionObject TableName))

ConnectionObject The return value of ADOLISP_ConnectToDB.

TableName The name of a table in the database (not case-sensitive)

Note: This function is often noticeably slow. It is usually preferable to get the information from the ADOLISP_FieldsPropertiesList variable after retrieving some data; see section 5.1 on page 12

4.3.2.2 Return Value

If it fails, ADOLISP_GetColumns returns NIL. If this happens, call (ADOLISP_ErrorPrinter) to print the error description(s).

If it succeeds, ADOLISP_GetColumns returns a list of lists, one sub-list for each column in the table. These sub-lists appear in the order that they physically appear in the table (“ordinal order”). Each sub-list begins with the name of the column, which is followed by five dotted pair lists of a property and its value, so you can use (cdr (assoc <PropertyName> ...)) to retrieve the value of a property from a sub-list (after using (cdr ...) to strip the field name from the front of the list). For example:

```
(( "DEVICE_NO"
  ("Type" . 129)
  ("DefinedSize" . 10)
  ("Attributes" . 104)
  ("Precision" . 255)
  ("Ordinal" . 1)
```

```

)
( "QUANTITY"
  ( "Type" . 5)
  ( "DefinedSize" . 0)
  ( "Attributes" . 120)
  ( "Precision" . 15)
  ( "Ordinal" . 2)
)
)

```

The Type property is an integer code for the data type of the field. The codes are explained in Appendix B, “Values and Meanings of the Field Type”, on page 18. In this case the “DEVICE_NO” field is a string value and the “QUANTITY” field is a double-precision floating-point value.

The DefinedSize property is the length, in bytes, of the field definition in the database (if the field is variable size, this is the maximum size). It is only meaningful for character fields; for other fields it will be 0.

The Attributes property is the sum of the appropriate values from appendix C, “Values and Meanings of the Field Attributes”, on page 21. For the “DEVICE_NO” field it is 64 + 32 + 8 (the field may contain nulls, you may write nulls to the field, the provider cannot determine if you can write to the field). For the “QUANTITY” field it is 64 + 32 + 16 + 8 (the field may contain nulls, you may write nulls to the field, the field is fixed length, the provider cannot determine if you can write to the field).

The Precision property is the maximum number of digits of a numeric property, or 255 if this property is not applicable.

The Ordinal property is the number of the column in the table, starting at 1. It is present so ADOLISP can sort on it to ensure that the list is in the same order as the columns in the table. You almost certainly will never need it.

In this example we could retrieve the type code of the “DEVICE_NO” field with:

```

(cdr (assoc "Type"
            (cdr (assoc "DEVICE_NO" ColumnList))))
)

```

5. ADOLISP Global Variables

5.1 *ADOLISP_DoNotForceJetODBCParsing – Jet Driver Property*

In previous versions of ADOLISP, when a user made a connection to a database using the Jet database engine, ADOLISP automatically set the “Jet OLEDB:ODBC Parsing” property to True, forcing the Jet driver to parse statements using ODBC rules rather than its own internal rules. This allows using standard SQL syntax in queries rather than Microsoft Access syntax.

However, it appears that it is not necessary to set this property in many circumstances. Often you can use standard SQL syntax without setting this property. In fact, setting that property causes

errors in some circumstances. I cannot describe the circumstances accurately; some users have reported problems when using an Excel 2000 spreadsheet as the database and setting this property.

As of version 2.4, you can set ADOLISP_DoNotForceJetODBCParsing to non-nil, for example:

```
(setq ADOLISP_DoNotForceJetODBCParsing t)
```

And ADOLISP will not set the Jet OLEDB:ODBC Parsing property when making a connection.

5.2 ADOLISP_FieldsPropertiesList – Properties of Retrieved Fields

Whenever ADOLISP_DoSQL executes a statement that retrieves data (that is, a select statement), it also sets a global variable to contain the properties of each field. The format is a list of lists, one sub-list for each field. The sub-lists are stored in the same order as the retrieved fields (so you can use either (assoc ...) or (nth ...) to retrieve a sub-list). Each sub-list starts with the name of the field (as it is named in the database) followed by six dotted pair lists of a property name paired with its value, so you can use (cdr (assoc <PropertyName> ...)) to retrieve the value of a property from a sub-list (after using (cdr ...) to strip the field name from the front of the list).

It is probably easiest to illustrate with an example. The following is the contents of ADOLISP_FieldsPropertiesList after executing “SELECT * FROM MYTABLE” on an Excel spreadsheet that contains two fields (“A” and “B”, in that order), each containing numbers:

```
((("A"  ("Attributes" . 116)
        ("DefinedSize" . 8)
        ("NumericScale" . 255)
        ("Precision" . 15)
        ("Type" . 5)
      )
  ("B"  ("Attributes" . 116)
        ("DefinedSize" . 8)
        ("NumericScale" . 255)
        ("Precision" . 15)
        ("Type" . 5)
      )
  )
)
```

The properties are not always defined *exactly* the same as the properties in the return value of ADOLISP_GetColumns.

The Attributes property is the sum of the appropriate values from appendix C, “Values and Meanings of the Field Attributes”, on page 21. In this case, both fields have an Attributes property of 116 (64 + 32 + 16 + 4), meaning that the field may be null, you can write null values to the field, the field contains fixed-length data, and the field can be written.

The DefinedSize property is the length, in bytes, of the field definition in the database (if the field is variable size, this is the maximum size).

The NumericScale property is either the number of digits stored to the right of the decimal point or, if the property is not applicable, 0 or 255 (depending on the provider).

The Precision property is the maximum number of digits of a numeric property, or 255 if this property is not applicable.

The Type property is an integer code for the data type of the field. The codes are explained in Appendix B, “Values and Meanings of the Field Type”, on page 18. In this case both fields are double-precision floating point values.

In this example we could retrieve the type code of field B with:

```
(cdr (assoc "Type"
            (cdr (assoc "B" ADOLISP_FieldsPropertiesList))
          )
)
```

5.2.1 Changes in Revision 2.31

The ActualSize property (the length, in bytes, of the value contained in the field) was removed from the ADOLISP_FieldsPropertiesList in revision 2.31. I found that asking about ActualSize when no records were retrieved (and some other conditions that I haven’t completely characterized) caused an automation error which cannot be trapped. I think it’s unlikely that anyone is using this item, but contact me if you need it back.

6. Examples

The files ADOLISP_Example.lsp and ADOLISP_test.mdb are a very basic example of connecting, retrieving data, modifying data, and disconnecting.

Create a folder named CAD in the root of drive C and save ADOLISP_test.mdb in C:\CAD. Note that this is a change from previous versions; ADOLISP cannot access a database in the root of the system drive in Vista.

Load ADOLISP_Library.lsp and ADOLISP_Example.lsp in AutoCAD 2000 or higher.

At the AutoCAD “Command:” prompt, type:

```
Example
```

And press the “Enter” key.

7. Connecting to Excel

Most of the information in this document, and all of the examples, relates to connecting to a Relational DataBase Management System (RDBMS) such as Microsoft Access. Many people want to connect to Microsoft Excel. This can be done but, since Excel is not an RDBMS, there is some extra setup required and there are limitations.

If you have difficulties connecting to Excel, try using the ADOLISP_DoNotForceJetODBCParsing variable as described in “ADOLISP_DoNotForceJetODBCParsing – Jet Driver Property” on page 12.

7.1 Setting up an Excel Spreadsheet

You must set aside an area in which the data is to be kept. This may be an area in a worksheet that also contains other information, but it is preferable to put the data in a worksheet of its own and put any formulas that refer to that data in another worksheet.

Excel does not report the type of the data to the database driver. The database driver must scan a part of the data area to determine whether the data is numbers, text strings, dates, or whatever. Although it does not affect the database driver, it is a good idea to format the columns with the appropriate data type just for visual presentation,

It is possible but very difficult to work with the data without assigning names to the columns. It is definitely best to format the first row of the data area as text and enter column names in that first row.

You may enter data in the appropriate places in Excel if you wish. Never enter formulas in the data area.

It is also possible to work with the data without assigning a name to the data area (see below). However, it is almost always best to assign a “table name” to the data area, which will be the name that you will use in your SQL statements where the table name goes. Highlight the column names and at least one row’s worth of the cells below them (including all data you have pre-entered). Pick “Insert” “Name” “Define” and enter a name. (Note: If you are using MDAC 2.5 or later, the named area will expand automatically when you add new data through SQL. If you are using MDAC before 2.5, you must highlight the entire area that you will be using for data, now and in the future. You can check on your version of MDAC using the Component Checker available at <http://www.microsoft.com/data>, or you can update your MDAC from the files at that site.).

(If you insist on not using a named range, you can refer to the data area by the sheet name followed by a \$ and enclosed in square brackets, such as “[Sheet1\$]”. If you do this, there may not be anything in any cells to the right of and above the data area. Using this syntax can lead to a lot of extra null rows being returned by a “select ...” statement.)

(You can even refer to individual cell ranges such as “[Sheet1\$A1:B4]”. This is dangerous because it often yields unexpected results. For example, if you ask for a range of cells that includes more than one row but does not include the column name row then the result is only the last row of the range and the column names are “F1”, “F2”,)

(Using non-named-range specification, that is the square-bracket specification of the affected cells, often leads to problems when inserting data; you get to insert data once but never again.)

7.2 Excel Limitations

You cannot delete a row. That is, any “delete from ...” SQL statement will fail. The best you can do is an “update ...” that sets all the data cells in that row to null.

7.3 Further information

See Microsoft Knowledge Base article Q257819, currently at <http://support.microsoft.com/default.aspx?scid=kb;en-us;Q257819>.

8. Hints and Tips

8.1 Database Table and Column Names

AutoCAD 2000 and ADO are much less picky about identifiers than previous versions. ADO is still sensitive to “regular” and “delimited” identifiers.

The names of databases, tables and columns are “identifiers”. An identifier can be “regular” or “delimited”. A regular identifier:

- Begins with a letter
- Contains only letters, numerals, and underscores (_). **No spaces!**
- Is not an SQL reserved word (see “ Appendix A SQL/92 Reserved Words” on page 18)

A delimited identifier is anything that is not a regular identifier. Whenever a delimited identifier appears in an SQL statement, it must be surrounded (delimited) by double quotation marks. Regular identifiers may, but do not have to be, surrounded by double quotation marks. For example, since “section” is an SQL/92 reserved word, the following will not work:

```
SELECT * FROM TABLE WHERE SECTION = 'FOT%'
```

but the following will:

```
SELECT * FROM TABLE WHERE "SECTION" = 'FOT%
```

8.2 Autonumber fields

Your database may contain fields in which sequential numbers (or something similar) are automatically created and inserted when a record is inserted. You do not have to care about this *except* when inserting records. If you have fields that the database *insists* on filling in by itself, then you must use the form of the INSERT statement that includes column names and you must not include the field or fields that the database will fill in:

```
INSERT INTO <table> (<field1>, <field2>, ...) VALUES (<value1>, <value2>, ...)
```

8.3 Setting Date and Time fields

Setting the value of date fields is sometimes difficult. Here are some examples that have been reported to work in various circumstances:

```
INSERT INTO <table> (<datefield>) VALUES (DateValue( 'April 3, 2002'))
```

INSERT INTO <table> (<datefield>) VALUES (TIMESTAMP 'April 3, 2002')

INSERT INTO <table> (<datefield>) VALUES (DATE 'April 3, 2002')

INSERT INTO <table> (<datefield>) VALUES (TIME 'April 3, 2002')

INSERT INTO <table> (<datefield>) VALUES (CAST('April 3, 2002' AS DATE))

Appendix A SQL/92 Reserved Words

ABSOLUTE	ACTION	ADD	ALL	ALLOCATE
ALTER	AND	ANY	ARE	AS
ASC	ASSERTION	AT	AUTHORIZATION	AVG
BEGIN	BETWEEN	BIT	BIT_LENGTH	BOTH
BY				
CASCADE	CASE	CAST	CATALOG	CHAR
CHARACTER	CHAR_LENGTH	CHARACTER_LENGTH	CHECK	CLOSE
COALESCE	COLLATE	COLLATION	COLUMN	COMMIT
CONNECT	CONNECTION	CONSTRAINT	CONSTRAINTS	CONTINUE
CONVERT	CORRESPONDING	COUNT	CREATE	CROSS
CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP		CURRENT_USER
CURSOR				
DATE	DAY	DEALLOCATE	DEC	DECIMAL
DECLARE	DEFAULT	DEFERABLE	DEFERRED	DELETE
DESC	DESCRIBE	DESCRIPTOR	DIAGNOSTICS	DISCONNECT
DISTINCT	DOMAIN	DOUBLE	DROP	
ELSE	END	END-EXEC	ESCAPE	EXEPT
EXCEPTION	EXEC	EXECUTE	EXISTS	EXTERNAL
EXTRACT				
FALSE	FETCH	FIRST	FLOAT	FOR
FOREIGN	FOUND	FROM	FULL	
GET	GLOBAL	GO	GOTO	GRANT
GROUP				
HAVING	HOUR			
IDENTITY	IMMEDIATE	IN	INDICATOR	INITIALLY
INNER	INPUT	INSENSITIVE	INSERT	INT
INTEGER	INTERSECT	INTERVAL	INTO	IS
ISOLATION				
JOIN				
KEY				
LANGUAGE	LAST	LEADING	LEFT	LEVEL
LIKE	LOCAL	LOWER		
MATCH	MAX	MIN	MINUTE	MODULE

MONTH

NAMES	NATIONAL	NATURAL	NCHAR	NEXT
NO	NOT	NULL	NULLIF	NUMERIC
OCTET_LENGTH	OF	ON	ONLY	OPEN
OPTION	OR	ORDER	OUTER	OUTPUT
OVERLAPS				
PARTIAL	POSITION	PRECISION	PREPARE	PRESERVE
PRIMARY	PRIOR	PRIVILEGES	PROCEDURE	PUBLIC
READ	REAL	REFERENCES	RELATIVE	RESTRICT
REVOKE	RIGHT	ROLLBACK	ROWS	
SCHEMA	SCROLL	SECOND	SECTION	SELECT
SESSION	SESSION_USER	SET	SIZE	SMALLINT
SOME	SQL	SQLCODE	SQLERROR	SQLSTATE
SUBSTRING	SUM	SYSTEM_USER		
TABLE	TEMPORARY	THEN	TIME	TIMESTAMP
TIMEZONE_HOUR	TIMEZONE_MINUTE	TO	TRAILING	TRANSACTION
TRANSLATE	TRANSLATION	TRIM	TRUE	
UNION	UNIQUE	UNKNOWN	UPDATE	UPPER
USAGE	USER	USING		
VALUE	VALUES	VARCHAR	VARYING	VIEW
WHEN	WHENEVER	WHERE	WITH	WORK
WRITE				
YEAR				
ZONE				

Appendix B Values and Meanings of the Field Type

The field type that is stored in the ADOLISP_FieldsPropertiesList is an integer code (a `DataTypeEnum`). The following table shows the meanings of the possible values.

Value	Name	Description
0	adEmpty	Specifies no value.
2	adSmallInt	Indicates a two-byte signed integer.
3	adInteger	Indicates a four-byte signed integer.
4	adSingle	Indicates a single-precision floating-point value.
5	adDouble	Indicates a double-precision floating-point value.
6	adCurrency	Indicates a currency value. Currency is a fixed-point number with four digits to the right of the decimal point. It is stored in an eight-byte signed integer scaled by 10,000.
7	adDate	Indicates a date value. A date is stored as a double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
8	adBSTR	Indicates a null-terminated character string (Unicode).
11	adBoolean	Indicates a boolean value.
14	adDecimal	Indicates an exact numeric value with a fixed precision and scale.
16	adTinyInt	Indicates a one-byte signed integer.
17	adUnsignedTinyInt	Indicates a one-byte unsigned integer.
18	adUnsignedSmallInt	Indicates a two-byte unsigned integer.
19	adUnsignedInt	Indicates a four-byte unsigned integer.
72	adGUID	Indicates a globally unique identifier (GUID).
129	adChar	Indicates a string value.
130	adWChar	Indicates a null-terminated Unicode character string.
131	adNumeric	Indicates an exact numeric value with a fixed precision and scale.
132	adUserDefined	Indicates a user-defined variable.
139	adVarNumeric	Indicates a numeric value.
200	adVarChar	Indicates a string value.
202	adVarWChar	Indicates a null-terminated Unicode character string.
203	adLongVarWChar	Indicates a long null-terminated Unicode string value.
204	adVarBinary	Indicates a binary value.

Appendix C Values and Meanings of the Field Attributes

Value	Name	Description
-1	adFldUnspecified	Indicates that the provider does not specify the field attributes.
2	adFldMayDefer	Indicates that the field is deferred—that is, the field values are not retrieved from the data source with the whole record, but only when you explicitly access them.
4	adFldUpdatable	Indicates that you can write to the field.
8	adFldUnknownUpdatable	Indicates that the provider cannot determine if you can write to the field.
16	adFldFixed	Indicates that the field contains fixed-length data.
32	adFldIsNullable	Indicates that the field accepts null values.
64	adFldMayBeNull	Indicates that you can read null values from the field.
128	adFldLong	Indicates that the field is a long binary field. Also indicates that you can use the AppendChunk and GetChunk methods.
256	adFldRowID	Indicates that the field contains a persistent row identifier that cannot be written to and has no meaningful value except to identify the row (such as a record number, unique identifier, and so forth).
512	adFldRowVersion	Indicates that the field contains some kind of time or date stamp used to track updates.
4096	adFldCacheDeferred	Indicates that the provider caches field values and that subsequent reads are done from the cache.
8192	adFldIsChapter	Indicates that the field contains a chapter value, which specifies a specific child recordset related to this parent field. Typically chapter fields are used with data shaping or filters.
16384	adFldNegativeScale	Indicates that the field represents a numeric value from a column that supports negative scale values. The scale is specified by the NumericScale property.
65536	adFldIsRowURL	Indicates that the field contains the URL that names the resource from the data store represented by the record.
131072	adFldIsDefaultStream	Indicates that the field contains the default stream for the resource represented by the record. For example, the default stream can be the HTML content of a root folder on a Web site, which is automatically served when the root URL is specified.
262144	adFldIsCollection	Indicates that the field specifies that the resource represented by the record is a collection of other resources, such as a folder, rather than a simple resource, such as a text file.

