ODBC Socket Server

REFERENCE MANUAL
Manual Version: 0.92
ODBC Socket Server Version: 0.92
Author: Team FXML

TABLE OF CONTENTS

INTRODUCTION

ODBC Socket Server is a Windows NT service that allows unprecedented ODBC database access across a multitude of different platforms. While other commercial packages normally cost in the thousands of dollars, ODBC Socket Server provides not only a rich feature set, but also all underlying code free of charge. With its XML based communication mechanism and open source heritage, ODBC Socket Server is the most customizable and feature-rich database access mechanism on the market today.

Team FXML, the developers of ODBC Socket Server, have years of database and XML experience. This product is yet another open source tool from the open source leader in Windows/Linux toolkits.

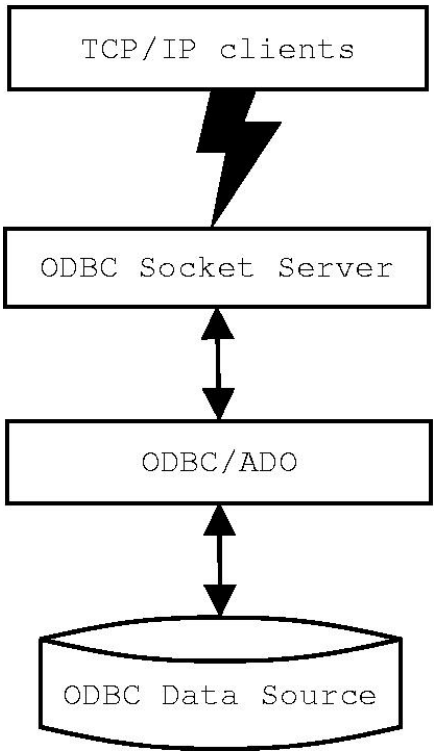
FEATURES

ODBC Socket Server contains many features that appeal to software developers and managers alike. These features are:

- Multithreaded, Windows NT service
- Empowers any TCP/IP client platform to execute any SQL command on any Windows ODBC datasource or OLEDB provider
- Uses a well-documented XML communication protocol to provide an extensible communication mechanism
- Fine grained security ensures only authorized clients can connect
- Highly configurable out of the box, infinitely configurable via source code manipulation
- Easy to debug: events are written to the NT Event log with an adjustable amount of detail
- Includes client libraries for any type of Windows, UNIX, or Linux system
- Completely free of charge with a very liberal license (Gnu Public License)
- Service and clients also run on Windows 98

Many people wonder why we are giving this product away for free. The reason is because we believe data warehouse access is just as important to application development as the development environment. One should not dictate the other and by preserving choice of servers and clients, ODBC Socket Server will ensure you have an extensible and sensible data access layer.

The following diagram illustrates the role of ODBC Socket Server in your enterprise:



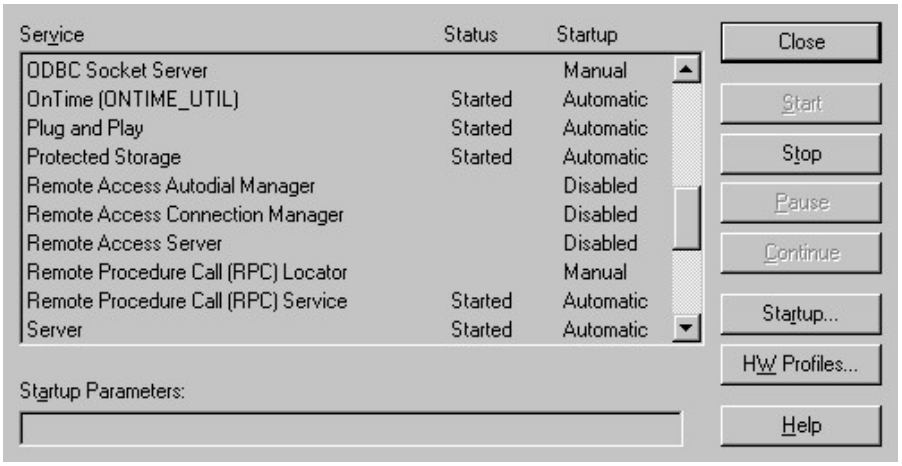
INSTALLATION ON WINDOWS NT

Before you install ODBC Socket Server, be sure you have the following software:

- Windows NT Version 4 or higher
- ADO libraries installed (included with higher service packs or the latest version can be downloaded from http://www.microsoft.com/data/)

ODBC Socket server is a simple Windows NT Service. To install it, perform the following steps:

1. Unzip the ODBC Socket Server application into a directory on your server.
2. Run the following command via the run option on the Windows NT start menu (or from a DOS box, whichever you prefer) <path to ODBCSocketServer.exe>\ODBCSocketServer.exe /Service
   For example, if you installed ODBC Socket server to c:\ODBC, you would run the command c:\ODBC\ODBCSocketServer.exe /Service
   If you get an error mentioning either a missing or ordinal problem with the ATL.dll file then copy your existing atl.dll from your winnt\system32 directory to a back-up location. Now copy the atl.dll version included in the ODBC Socket Server distribution (in the base directory) into your winnt\system32 directory. Last, register this dll via the command "regsvr32 c:\winnt\sytem32\atl.dll" (assuming that is where your atl.dll is located). You should now be able to register ODBCSocketServer.exe via the "ODBCSocketServer.exe /Service" command with no problems.
   Running this command registers ODBC Socket Server with Windows NT as a service
3. To enter information about ODBC Socket Server into the database for configuration, open the Windows NT Explorer and navigate to the directory where you unzipped ODBC Socket Server. In this path there is a file called "Registry.reg". Double-click on this file from NT explorer to enter the default configuration settings into your registry. A precise description of these settings is provided later on in this documentation.
4. Using the control panel, open up the services dialog. You should see the ODBC Socket Server present in the list of services:



5. To start ODBC Socket Server, simply press Start. If you would like ODBC Socket Server to automatically start each time Windows NT starts, press the Startup button and select the "Automatic" option.
6. Please skip to the "Testing Your Installation" section below to test your ODBC Socket Server installation

INSTALLATION ON WINDOWS 98

Installing ODBC Socket Server on Windows 98 is much more condensed than Windows NT. However, there are no ODBC Socket Server binaries for Windows 98 distributed with ODBC Socket Server. Therefore, you must rebuild ODBC Socket Server from source (found in the server_source directory) to generate a Windows 98 binary.

Windows 98 binaries are not included in the base ODBC Socket Server because Windows 98 is not the optimal operating environment for ODBC Socket Server. However if Windows 98 is your only option, use the following steps to install ODBC Socket Server

1. Unzip ODBC Socket Server into a directory on your Windows 98 machine
2. In the server_source directory, open up the ODBCSocketServer.dsp file using Microsoft Visual C++. Compile a new binary using the build conifguration "Win32 Release".
3. Simply run the resulting ODBCSocketServer.exe binary from Windows Explorer by double-clicking on it. To run the service each time the machine is started, place a shortcut to ODBCSocketServer.exe in the Startup menu.

TESTING YOUR INSTALLATION

To test ODBC Socket Server, you must first create an ODBC System Data Source on your Windows NT Machine or use an OLEDB provider. To create an ODBC System Data Source, follow these steps:

1. On the Start menu, point to Settings, and then click Control Panel.
2. Double-click ODBC.
3. Click System DSN and then click Add.
4. Click the type of data source you would like to add; then click Finish.
5. Complete the steps in the Create a New Data Source to SQL Server Wizard.

Once an ODBC data source has been created, you can now test a query. To test the query, we will use the sample client application the came with ODBC Socket Server. To use the test client application, perform the following steps:

1. Register the ODBC Socket Server Client COM DLL. To do this, run the following command via the run option on the Windows NT start menu (or from a DOS box, whichever you prefer) regsvr32.exe <path to ODBCSocketServer.exe>\clients\COM\ODBCClient.dll
   For example, if you installed ODBC Socket server to c:\ODBC, you would run the command regsvr32.exe c:\ODBC\clients\COM\ODBCClient.dll
   In case your path is not setup correctly, regsvr32.exe lies in your Windows system32 directory.
2. Start the COM sample application, SockTest.exe (in the same directory as

ODBCClient.dll).

3. Enter the IP address, a valid ODBC connection string, and some SQL to execute. In the available text boxes. Examples of valid ODBC connection strings include "DSN=Pubs;UID=smith;PWD=muffins;" provided you have created an ODBC data source named Pubs on the ODBC Socket Server machine. Examples of valid SQL include "SELECT * FROM authors"
4. Click on the ExecSQL button to execute this SQL.
5. The results of your query should now appear in the results pane. If you don't see any results or the application times out, check the NT event log on both the ODBC Socket Server machine and the machine you are running this test application on. Be sure to check the Application event log!
   The error messages in this log should help you diagnose your problem. If you are still having trouble, e-mail the authors at fxml@excite.com with details of your problems.

ODBC SOCKET SERVER CLIENT TOOLS

The purpose of this section is to provide the reader with an introduction to the array of client tools provided with ODBC Socket Server. All client tools include well-documented source and also include a small, simple application that demonstrates the client tool.

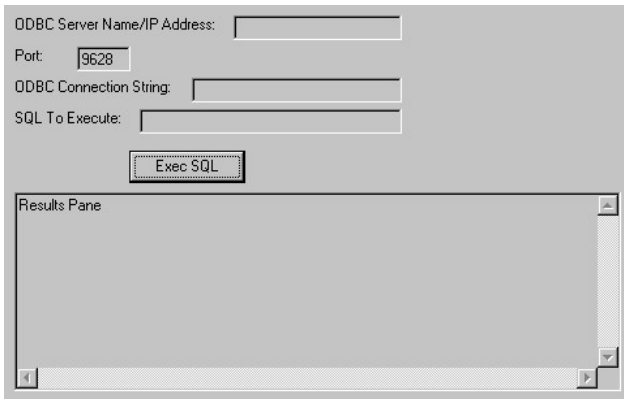
ODBC Socket Server ships with three different client utilities:

1. A COM object and Visual Basic application that execute remote queries on a server and display the results in a text box. The real meat of this client is the COM object, which can be embedded in any Microsoft application to provide ODBC Socket Server functionality. The COM object will execute SQL on an ODBC Socket Server and return the XML result string.
2. A Perl module and cgi script that demonstrate how to build an Internet gateway to ODBC Socket Server. Since these scripts were developed on Linux, they demonstrate the multi-platform capabilities of ODBC Socket Server and should allow any perl developer to easily embed Microsoft ODBC databases into their Linux or UNIX web application. The main method of the perl module will execute a SQL statement on an ODBC Socket Server and return the corresponding XML string.
3. A C++ class and corresponding test application that were developed on Linux (and should work without any source changes on any UNIX distribution). The C++ class, like the aforementioned perl client, demonstrates how to call a remote Microsoft ODBC database from a UNIX platform and wraps this functionality with an easy to use interface.
4. A PHP class and script demonstrating its use. This script uses the excellent PHP socket functions to connect and query an ODBC Socket Server machine. This script was developed and tested on Linux using PHP3, though it should work on other platforms and PHP 4 without modification.
5. A Python class and script demonstrating its use. This class was generously developed and contributed by Marc Risney (marc.risney@blazesoft.com) and Jochen Knuth (J.Knuth@ipro.de). It uses Python socket classes to interface with ODBC Socket Server.

COM INTERFACE

The COM interface consists of a COM object programmed in C++ (using ATL) and uses the Winsock sockets interface.

Also included is a Visual Basic application that demonstrates how to use the COM object. Its startup screen is as follows:



The actual COM interfaces exposes the following attributes:

Name	Type	Description
ConnectionString	BSTR	ODBC connection string to use
DebugLevel	short	Level of debug information to output to event log. The following values are used: 1- report as much information as possible (not as verbose as it sounds) 2- report only socket errors and fatal errors (recommended) 3- report only fatal system errors
HostName	BSTR	Hostname or IP address of the ODBC Socket Server to connect to
Port	short	Port to connect to. Default is 9628.
SocketTimeout	short	Timeout in seconds while waiting for the server response. Default is 30.

The only method exposed is:

Name	Parameters	Returns	Notes
ExecSQL	sSQL - SQL string to execute	XML result string (BSTR)	Executes SQL on the server and returns the XML result string.

PERL INTERFACE

The Perl module for ODBC Socket Server is a simple module that uses perl sockets to communicate with the ODBC Socket Server.

The module differs from the other provided clients in that it provides no attributes, just one method described below:

Name	Parameters	Returns	Notes
ExecSQL	Server - the server to connect to (IP address or name) ConnectionString - the ODBC connection string to use SQL - the SQL command to execute	XML result string	Connects via sockets to the server and executes the SQL, returning the XML that is ready for parsing.

A related tool you will want to use is the Perl::XML parser, a Perl parser based on James Clark's EXPAT XML parser. A great description of this module is at http://www.xml.com/xml/pub/98/09/xml-perl.html

Team FXML followers will be familiar with EXPAT as parsing module shipped with fXML. EXPAT is an excellent XML parser that has been perfectly merged into the perl language.

As a sample Perl application, a quick CGI fronted to this module has been created. This application can be run from a web browser by pointing the web browser to the supplied index.cgi file on a web server (configured so that the server can run cgi scripts).

UNIX C++ CLASS INTERFACE

The UNIX C++ interface was designed to be a method and attribute-perfect port of the Windows COM interface. If you are a developer who has a little cross-platform development experience, it is interesting to contrast the socket methods of the COM interface and this interface. In some areas they are very similar, in others they are quite different. Also, please do not let the word "port" deceive you! This class was ground-up native Linux code, developed under K-Develop (project file included), an incredible IDE that provides many advanced features. K-Develop can be downloaded at http://www.kdevelop.org/

The C++ class has the following public attributes:

Name	Type	Description
nPort	short	Port to connect to. Default is 9628.
nSocketTimeout	short	Timeout in seconds while waiting for the server response. Default is 30.
sConnectionString	std::string	ODBC connection string to use
sHostName	std::string	Hostname or IP address to connect to

The C++ class has one public method:

Name	Parameters	Returns	Notes
ExecSQL	sSQL - SQL string to execute	XML result string	Executes SQL on the server and returns the XML result string.

PHP INTERFACE

The PHP interface takes the form of a class. It (like the other clients) queries an ODBC Socket Server and returns the XML string. The permier XML parser for PHP, EXPAT, must be compiled into PHP though. To compile PHP with XML parser support you must use the --with-xml option. Full documentation on using XML with PHP is available at the PHP site, http://www.php.net/
The interface of the class is almost identical to the C++ class.

The PHP class has the following public attributes:

Name	Type	Description
nPort	short	Port to connect to. Default is 9628.
sConnectionString	string	ODBC connection string to use
sHostName	string	Hostname or IP address to connect to

The PHP class has one public method:

Name	Parameters	Returns	Notes
ExecSQL	sSQL - SQL string to execute	XML result string	Executes SQL on the server and returns the XML result string.

PYTHON INTERFACE

The Python class was generously developed and contributed by Marc Risney (marc.risney@blazesoft.com) and Jochen Knuth (J.Knuth@ipro.de). It uses Python socket classes to interface with ODBC Socket Server. It has uses a constructor to set host and connection string information

Name	Parameters	Returns	Notes
ODBCSocketServer (constructor)	Server - the server to connect to (IP address or name) ConnectionString - the ODBC connection string to use	ODBC Socket Server object	The constructor intializes the server and connection string.
ExecSQL	SQL - The SQL to execute	The XML result from the ODBC Socket Server	Executes SQL on the server and returns the result

CONFIGURING ODBC SOCKET SERVER

The purpose of this section is to give you an overview of the out-of-the-box configuration options of ODBC Socket Server.

ODBC Socket Server uses the Windows NT registry to load its special settings. Although it is possible to run ODBC Socket Server without the registry settings, this practice is not advised as it will result in ODBC Socket Server emitting the maximum amount of status information to the NT Event Log and clients being able to connect from any host.

The registry key "HKEY_Local_Machine\Software\ODBC Socket Server" contains all ODBC Socket Server configuration options. In this key, the following string values are present:

Registry Value	Description
DebugLevel	The amount of event information that is reported to the NT Event Log. The following values are used: 1- report as much information as possible (not as verbose as it sounds) 2- report only socket errors and fatal errors (recommended) 3- report only fatal system errors The default value is 1. The recommended value is 2.
IP.Allow	A list of the allowed client IP Addresses. Format for IP.allow is as follows: A semicolon-delimited list of IP Addresses with "*" used as a wildcard. Default value is "*" (allow all clients to connect). Example: 127.0.0.1;10.4.1.*; - Allow all localhost connections - Allow connections from any machine that has an IP address starting with 10.4.1.
IP.Deny	A list of client IP Addresses that are not allowed to connect. Format for IP.deny is the same as IP.allow. If IP.allow is "*", then all clients are allowed except those specified in IP.deny. If IP.allow is a list of clients, then no clients are allowed except those specified in IP.allow. Example: IP.allow = 127.0.0.1; 10.4.1.* IP.deny is empty Machine 10.4.1.24 is allowed to connect. Machine 10.4.2.25 is NOT allowed to connect.
MaxThreadCount	Maximum number of threads (or socket connections) allowed active at one time. Default number is 5.
SocketTimeout	The time in seconds to wait for socket data to arrive after a socket connection has been established. The default value is 15.

SQLTimeout	The time to wait for SQL communication to occur in seconds before timing out. The default value is 15.
Port	The registry port that ODBC Socket Server listens for connections on. The default port is 9628.
UseCDATA	Integer value (0 or 1). Specifies whether XML CDATA tags should be used when sending data back from the server to the client. The default value is false, 0.
UseMSDTD	Integer value (0, 1, or 2). Specifies what data format is emitted from ODBC Socket Server. For small amounts of data, use 0, ODBC Socket Server's native format. For faster results in a similar format, use 2. For the fastest results, use the native Microsoft format, 1. The recommended value for large data sets is 1. For small data sets, 0. The PHP and Perl samples parse output using value 0. Default is 0.

Of note is that the information in the registry is re-read each time a socket connection is established. Therefore changing this information does NOT force you to restart the ODBC Socket Server.

COMMUNICATION PROTOCOL

ODBC Socket Server uses an XML-based TCP/IP stream socket protocol to communicate between a client and server. The advantages to this protocol are that any client can connect and interact with the TCP/IP server. Macintosh, Linux, BSD, VAX, if a client supports TCP/IP, it can talk to ODBC Socket Server. Also, the plethora of XML parsers available for all of these platforms means that parsing ODBC Socket Server data is very straightforward.

The disadvantages to using XML are that communication is more verbose than it would be if a straight binary stream was used, and the exchange of clear text data exposes a potential security risk if communication is taking place via a foreign network.

Future releases of ODBC Socket Server may include encryption as a transport option. Using encryption would address the security concern, however as will be discussed in the security section, exchanging ANY unencrypted data through a foreign network is generally discouraged.

The default ODBC Socket Server communication protocol is very straightforward. The ODBC Socket Server client makes a request to the server, and the server responds to this request and then terminates the connection once all data has been sent. The data that is sent has the following format:

Client Request DTD:

```
<!ELEMENT request       (connectionstring, sql)>
<!ELEMENT connectionstring (#PCDATA)>
<!ELEMENT sql (#PCDATA |  #CDATA)>
```

Example:

```
<?xml version="1.0"?>
<request>
<connectionstring>DSN=pubs;UID=jsmith;PWD=test;</connectionstring>
<sql>select * from party where id = 3</sql>
</request>
```

The most important thing to note is that the sql element is denoted here as PCDATA. However it does not have to be PCDATA, CDATA is also acceptable. This way, if you wanted a query such as "SELECT * FROM party WHERE id > 5" you could send:

```
<sql>SELECT * FROM party WHERE id &gt; 5</sql>
```

or

```
<sql><![CDATA[SELECT * FROM party WHERE id > 5]]></sql>
```

and either would be accepted.

Server Response DTD:

```
<!ELEMENT result (error | row*)>
<!ATTLIST result
        state    (success |  failure)
>
<!ELEMENT row (column*)>
```

```
<!ELEMENT column (#PCDATA |  #CDATA)>
<!ATTLIST name (#PCDATA)
>
<!ELEMENT error (#PCDATA)>
```

Example of successful query:

```
<?xml version="1.0"?>
<result state="success">
<row>
<column name="client_name">John Smith</column>
<column name="date_added">1999-04-01 0:0:0</column>
</row>
<row>
<column>Peter Ross</column>
<column>1994-12-14 15:45:35</column>
</row>
</result>
```

Example of failed query:

```
<?xml version="1.0"?>
<result state="failure">
<error>ODBC reports data source name not found</error>
</result>
```

The example of the successful query is more interesting here. What is returned is an XML representation of an array, with each row containing the data from the recordset row that was returned, if any. The first set of column data returned has a name attribute, which specifies the name of the column.

A note about data types. The following data types are supported by the Socket Server: String, Number, Float/Real/Double etc, Date, Currency, Memo (long text), and Binary (BLOB). Date data is returned in the format YYYY-MM-DD HH:MM:SS. For string and memo data, by default the data is returned in valid PCDATA format. This constraint means that if the string "5 > 4 > 3" is to be returned, it would be returned as "5 > 4 > 3". However this behavior can be overridden and CDATA can be returned instead. This is done via the registry setting UseCDATA. If UseCDATA is true, then in the above example the string "<![CDATA[5 > 4 > 3]]>" would be returned instead.

This simple means of communication is one of the aspects of ODBC Socket Server that aspiring developers may want to customize. For instance, if you are dealing with large amounts of data being transferred from one Windows platform to another, you may want to implement a quick binary encoding routine to ensure that not an overwhelming amount of data gets exchanged. Keep in mind that doing this encoding will probably rule out connections from UNIX or Macintosh clients!

Astute readers will note that the data format described in this section is for the default ODBC Socket Server DTD. If data is required to be returned using the Microsoft ADO DTD, then the registry setting "UseMSDTD" can be set to 1. This DTD is not documented in this document and Team FXML has no control over future changes to the DTD. Therefore

it is recommended that you use the ODBC Socket Server DTD unless you are exchanging very large amounts of data (where the MS DTD may be more efficient) or you have legacy applications expecting the MS DTD data format.

The other output format ODBC Socket Server provides is similar to the original format with the difference that column titles appear for each row and NULL values are not transmitted.

DEVELOPMENT INFORMATION

This section is meant to be a reference for developers who wish to examine or modify the source code of this application so they can very quickly get up to speed with the specifics of the ODBC Socket Server.

ODBC Socket Server was created using Microsoft Visual C++ 6. It was implemented as an ATL service which uses the Winsock layer for communication. The Standard Template Library is used for most all string manipulation, and MFC is not used. Because of these design decisions, this service is very lightweight and fast.

ADO is used for database connections, and it is highly recommended that the latest OLEDB provider is used for your datasource as tremendous speed gains have been made over the past year with the newest releases of data providers.

The main application runs from the ODBCSocketServer.cpp file. Upon starting the service, a thread is spawned that instantiates the sockets layer and then calls procedure DoWinsock in file main-server.cpp. It is in this file that the majority of work is done. Important procedures are ReadAndWriteToSocket (which is responsible for most socket communication) and VariantToXML (which takes the variant recordset returned and converts it to XML). Note that for the alternate data outputs, the methods MSDTDToOurDTD and RecordsetToMSDTD are used instead of VariantToXML.

To avoid the requirement of installing Internet Explorer, Team FXML has used their own parser, fXML as the XML parsing component of this application. fXML is an open source, multiplatform XML toolkit that has wrappers for common XML parsers as well as its own internal parser. fXML can be found on the Internet at :http://www.geocities.com/ ResearchTriangle/System/5640/

It is statically compiled into ODBC Socket Server and all fXML source code used in ODBC Socket Server is provided as part of the ODBC Socket Server project. No additional code is necessary to get ODBC Socket Server to install.

SECURITY

Any TCP/IP application must enforce tight security and ODBC Socket Server is no exception. Security in ODBC Socket Server is provided on several levels. Unauthorized access to ODBC Socket Server is provided via the IP.allow and IP.deny registry entries. Correctly configuring these entries can ensure that only valid hosts connect to ODBC Socket Server.

If your data is being passed through potentially insecure networks, then it is imperative that you use a combination of an Internet fire-wall and a Virtual Private Network toolkit such as Free S/WAN (http://www.xs4all.nl/~freeswan/). No matter what data access toolkit you use, any data passing over an insecure network must be encrypted to prevent unwanted data access.

ROADMAP AND CHANGELOG

Future versions of the ODBC Socket Server will include the following features:

- Built in encryption/encoding technology
- Interfaces for ODBC and DBI
- KDE compliant SQL query analyzer
- Any other suggestions that we receive!

There is no timeframe for these features to be implemented.

Version 0.92 of ODBC Socket Server had the following changes:

- New Python client
- Windows 98 documentation included
- Fixed small state/status problem in main service
- Fixed registry file

Version 0.91 of ODBC Socket Server incurred the following changes:

- Added correct version of ATL.dll to the base distribution
- Added PHP client to the base distribution
- Removed server code to adjust recordset cache that was not supported by all ADO providers
- Fixed threading bug in service
- Fixed buffer overrun in event logging
- Small stability enhancements to main service and XML parser
- Added extra output format

Team FXML wishes you the best with ODBC Socket Server and we hope you find this service as useful as we have. Thank you for downloading, and enjoy!

CREDITS

Team FXML wishes to thank the following people for helping to develop and improve ODBC Socket Server:
The Python class was generously developed and contributed by Marc Risney (marc.risney@blazesoft.com) and Jochen Knuth (J.Knuth@ipro.de).
Steve Philp (sphilp@advancepkg.com) for his helpful debugging of the main service.