eGenix.com

mxODBC
Connect

Database Interface
for Python

Contents

Contents

1. Introduction

mxODBC has proven to be the most stable and versatile ODBC interface available for Python. It has been in use by many Python users and companies for years and is actively maintained by eGenix.com to meet the requirements of modern database applications, which our customers have built on top of mxODBC.

This manual will give you an in-depth overview of mxODBC Connect, the new networked client-server edition of mxODBC, providing ease of configuration, ease of deployment and scalability for all your mxODBC applications. It is written as technical manual, so background in Python and database programming is needed.

1.1 Technical Overview

mxODBC Connect allows your existing mxODBC based applications to access ODBC databases over a TCP/IP network and enables you to implement load balancing, fail-over, virtualisation and related technologies for your application.

mxODBC Connect consists of a stand-alone server and client packages which emulate the mxODBC API on the client side.

The mxODBC Connect Server is available for Windows, where it runs as Windows service, and on Unix platforms, where it can be deployed as daemon process.[1]

The client package emulates the native mxODBC API, so you can continue to use your application code when porting from the stand-alone version of mxODBC to mxODBC Connect. Furthermore, mxODBC Connect will allow you to port your application to platforms which were previously not supported by mxODBC due to limited availability of ODBC drivers.

---

[1] For the list of available platforms, please see the *eGenix.com website*.

1.2 Security

Unlike many ODBC drivers, mxODBC Connect comes with optional support for SSL based encryption of all communication, making it possible to send queries and data over public or otherwise unsafe networks.

Security can further be enhanced by enabling certificate verification, which will lower the risk introduced by the possibility of stolen database passwords or security holes in the database server or the underlying network architecture.

> **Note:**
> Only the communication between the mxODBC Connect Client and Server is encrypted. The ODBC driver used by the mxODBC Connect Server may still send unencrypted data and queries over the network. Please consult your ODBC driver documentation for details.

You can minimize this risk by installing the mxODBC Connect Server directly on the database server, e.g. the Windows machine running SQL Server. In most cases, the ODBC driver of the target database will then use lower level interfacing techniques such as shared memory, pipes or domain sockets to communicate with the database kernel, so that no communication is sent over the network.

1.3 Scope

This manual only explains features and configuration of the mxODBC Connect product.

Please refer to the *mxODBC User Manual* for mxODBC and Python DB API 2.0 specific details. The *mxODBC User Manual* contains all the needed details to develop against the mxODBC API exposed by the mxODBC Connect product.

2.   mxODBC Connect Server Installation

The mxODBC Connect product consists of a stand-alone server component and client packages for various platforms. The installers for both components are distributed separately.

The mxODBC Connect Server needs to be installed and configured only on the machine that has the ODBC driver you wish to use. This will typically be the database server itself.

The mxODBC Connect Server is a stand-alone product and comes with its own Python run-time, so you don't need to install Python separately on the server. Existing Python installations on the server are not modified in any way by the mxODBC Connect Server.

2.1   Upgrading mxODBC Connect Server

This section addresses server side changes between releases. For mxODBC Connect Client changes, please see section 3.1 Upgrading mxODBC Connect Client.

2.1.1   Upgrading from 2.0 to 2.1

Changes to user authentication

In mxODBC Connect, we changed the way passwords are stored in the server's authorized-users.txt file in order to make password storage more secure. The file now stores salted SHA-256 password hashes instead of the MD5 hashes used in version 2.0 and earlier. As a result, version 2.0 authorized-users.txt files will no longer work with version 2.1.

If you are using the user authentication feature of mxODBC Connect, please create a new authorized-users.txt file using the included `password-tool`.

Please see section 2.4.5 Configuring User Authentication for details.

> Note that using **certificate access authentication is recommended** over user authentication using username and password. It is far more secure than the application protocol level user authentication, since it is applied at the network protocol level. Section 2.4.4 Configuring Certificate Based Authentication explains how this is setup.

Update to the mxODBC 3.3 API

mxODBC Connect uses mxODBC 3.3 on the server and exposes almost all new features on the client side as well. Please see please see section 3.1 Upgrading mxODBC Connect Client for details.

This should not require any changes on the server side, except perhaps a possible update or use of the new `connection_cursortype` server configuration parameter. Please see the section 2.1.2 Upgrading from 2.0.x to 2.0.4 for details.

ODBC Driver/Manager Compatibility Enhancements

unixODBC

- mxODBC Connect Server is now built against **unixODBC 2.3.2** on Linux.

DataDirect

- **Updated the DataDirect binding to version 7.1.2** of the DataDirect ODBC manager on Linux.

Oracle

- Added work-around for **Oracle Instant Client** to be able to use integer output parameters.

- Added a work-around for **Oracle Instant Client** to have it return output parameters based on the input placeholder Python parameter types. It would otherwise return all parameters as strings.

- Disabled a test for **Oracle Instant Client** which tries to set a pre-connect connection option for timeouts, since the ODBC driver segfaults with this option.

MS SQL Server

- mxODBC Connect Server now defaults to 100ns connection.timestampresolution for **MS SQL Server 2008 and later**, and 1ms resolution for **MS SQL server 2005** and earlier. This simplifies interfacing to SQL Server timestamp columns by preventing occasional precision errors.

- Tested mxODBC Connect Server successfully with new **MS SQL Server Native Client 11 for Linux**. Unicode connection strings still don't work, but everything else does.

- Added documentation on how to **use Kerberos with mxODBC and SQL Server** for authentication on both Windows and Linux to the *mxODBC User Manual*.

- Added note about problems of the **FreeTDS ODBC driver** dealing with TIME and DATE columns to the to the *mxODBC User Manual*.

Sybase ASE

- Added work-around for the **Sybase ASE ODBC driver**, which doesn't always pass back NULL correctly to mxODBC Connect Server on 64-bit Unix systems.

- Changed the variable type binding mode default for the **Sybase ASE ODBC driver** from Python type binding to SQL type binding, which resolves issues with e.g. the Unicode support for that driver.

- Added note about a segfault problem with the **Sybase ASE 15.7 ODBC driver** which is caused by the driver corrupting the heap.

IBM DB2

- Added work-around for the **IBM DB2 ODBC driver**, which doesn't always pass back NULL correctly to mxODBC Connect Server on 64-bit Unix systems.

PostgreSQL

- Added work-around to force Python type binding for the **PostgreSQL ODBC drivers**. More recent versions of the driver report supporting SQL type binding, but they don't implement it.

- Added work-around to have **PostgreSQL ODBC drivers** properly work with binary data for BYTEA columns.

MySQL

- mxODBC Connect Server now supports native Unicode with the recent **MySQL ODBC drivers** - provided you use the Unicode variants of the drivers.

- Changed the default binding mode for **MySQL ODBC drivers** to Python type binding. This works around a problem with date/time values when talking to MySQL 5.6 servers.

2.1.2   Upgrading from 2.0.x to 2.0.4

New connection_cursortype server configuration parameter

In version 2.0.4 of the mxODBC Connect Server, we have added a new configuration setting to the `[Connection]` sections called `connection_cursortype`.

This allows you to pre-configure the mxODBC `connection.cursortype` to a fixed value without having to change the client side application.

Enhance MS SQL Server and IBM DB2 Fetch Performance

With this new server side setting you can adjust the used ODBC cursor type easily. Specifically for Microsoft SQL Server and IBM DB2 using forward-only cursors instead of the default static cursors is strongly advised - unless you have a need for static cursors. Please see the *mxODBC User Manual and Reference Guide* for details on the various cursor types.

Here's an example of how to change your server side configuration to benefit from the enhanced performance using forward-only cursors:

```
[Connection_Office]
    interface = 192.168.0.12
    netmask = 255.255.255.0

    # Use the faster forward-only cursors
    connection_cursortype = SQL.CURSOR_FORWARD_ONLY
```

With this setting, all client applications connecting to the given server connection will automatically benefit from the faster forward-only cursors.

Please note that while forward-only cursors provide better performance, they may also exhibit unwanted behaviour due to result sets not being scrollable anymore.

2.1.3 Upgrading from 1.0 to 2.0

Windows Service Changes

If you have version 1.0 of the mxODBC Connect Server running on a machine, please stop the server prior to upgrading. Not doing so can lead to error messages during the installation of version 2.0, e.g. due to timing and lock issues.

The version 2.0 installer will try to shutdown the 1.0 installation prior to continuing with the installation, but since this is done asynchronously by Windows, it is possible that the 1.0 server hasn't completely shut down in time for the version 2.0 installer to proceed with the installation.

If you get error message like "service marked for deletion" during an upgrade, please follow these steps:

1. manually uninstall the version 1.0 mxODBC Connect Server using the Windows control panel,

2. restart the machine and then

3. proceed with the version 2.0 mxODBC Connect Server installation.

Configuration File Changes

When upgrading from 1.0 to 2.0, you can leave the configuration files in place. Version 2.0 of the server still knows about the 1.0 configuration settings and will apply them in the same way.

Unlike version 1.0, version 2.0 is now using a single port to implement SSL and plain-text communication. As a result, the configuration setting `using_ssl` was replaced with `require_ssl` and `allow_ssl`. This offers more flexibility in the setup.

You can still use a two port configuration, if you like, but the default port no longer switches from 6632 to 6633 in case you enable SSL in the connection section.

Security Related Changes

mxODBC Connect Server 2.0 uses SHA1 digest values instead of MD5 digest values. This change will increase security of the client certificate checks.

Affected are the server configuration options `client_certificate_digest` and `client_certificate_digest_file`. Both now require SHA1 HEX digests instead of MD5 HEX digests as was needed for mxODBC Connect Server 1.0.

Network Related Changes

For the version 2.0 of the server, we have registered the port 6632 used by mxODBC Connect with IANA (as *mxodbc-connect* service).

Since assigned ports are a rare resource, port 6633 is no longer used by the server per default. However, you can still configure the server to use this port, if needed.

Port 6632 can now be used for both SSL and plain-text communication. It is even possible to have a mixed setup where some clients use plain-text and others use SSL communication over that port.

mxODBC Feature Changes

mxODBC Connect Server uses the new mxODBC 3.2 version on the server side, which provide better compatibility with current ODBC drivers and also include a number of new features compared to the older mxODBC 3.0 version included in mxODBC Connect Server 1.0.

Please see the *mxODBC User Manual and Reference Guide* for details on the mxODBC ODBC driver support enhancements.

2.2 mxODBC Connect Server Installation on Windows

The mxODBC Connect Server installer for Windows includes support for the *Microsoft ODBC Manager*, so you can use all available Windows system tools to configure your ODBC data sources.

2.2.1 Prerequisites

- You will need ODBC drivers for all database servers you wish to connect to. Windows comes with a very complete set of such drivers and most database vendors also provide Windows ODBC

drivers for their databases, but if you can't find the driver you are looking for have a look at section 8 *Hints & Links to other Resources*.

- Configure all your databases in *Microsoft ODBC Manager* as **System Data Sources**. System Data Sources are preferred, since any local user can access them. Note that the mxODBC Connect Server runs as the local system user by default, not as the administrative user you used for installation.

- On **64-bit Windows** systems, please make sure that you are configuring the right ODBC manager variant for the version of mxODBC Connect you have installed. Windows provides the ODBC manager as 32-bit version and 64-bit version, each with a separate list of data sources. If you install the 64-bit version of mxODBC Connect, the default ODBC manager from the C:\Windows\System32\ directory should be used. If you are using the 32-bit version of mxODBC Connect on Windows x64, please use the 32-bit version instead. You can usually find it in the C:\Windows\SysWOW64\ directory. After installation of the mxODBC Connect Server, you will find an entry "Start the ODBC Manager" in its start menu which will point to the right manager for your installation.

- Please make sure that all your data sources are accessible. This can be tested in the *Microsoft ODBC Manager* at the end of the DSN configuration wizard.

- The mxODBC Connect Server does not require any version of Python to be installed, since it comes with it's own Python runtime and the required set of libraries. The server will not interfere or depend on any existing Python installation on your server.

- You may need *administrator* privileges on Windows XP/2003 and later to successfully complete the installation or un-installation process.

2.2.2 Procedure

> **Note:**
> Please run the installer/uninstaller as *administrator* on Windows XP/2003 and later.

Please **uninstall any existing version of mxODBC Connect Server**, if you have a previous version installed (see section 2.2.3 for details).

After you have downloaded the Windows installer of the `egenix-mxodbc-connect-server` distribution, double-click on the executable to start the installation process. Depending on your OS version, you may need to click through a user account control (UAC) security dialog to proceed. Then follow the instructions of the installer.

If you run into a problem during the installation process, please consult section 7. Troubleshooting.

> **Note:**
> You have to provide a valid mxODBC Connect Server license to use the service. It's possible to install without a valid license file, but the service won't start.

Step-by-step Installation

The following screenshots demonstrate a typical installation. Please note that the screens may look different depending on your version of the installer and OS.

After you double click on the installer, the installation wizard starts up:
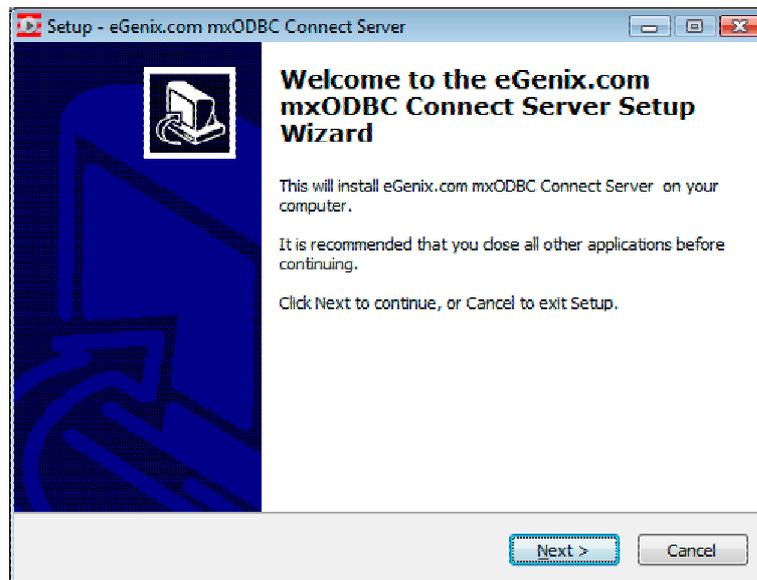
Clicking "Next" will take you to the next screen. "Cancel" aborts this installation process.

First you have to accept the license agreement:
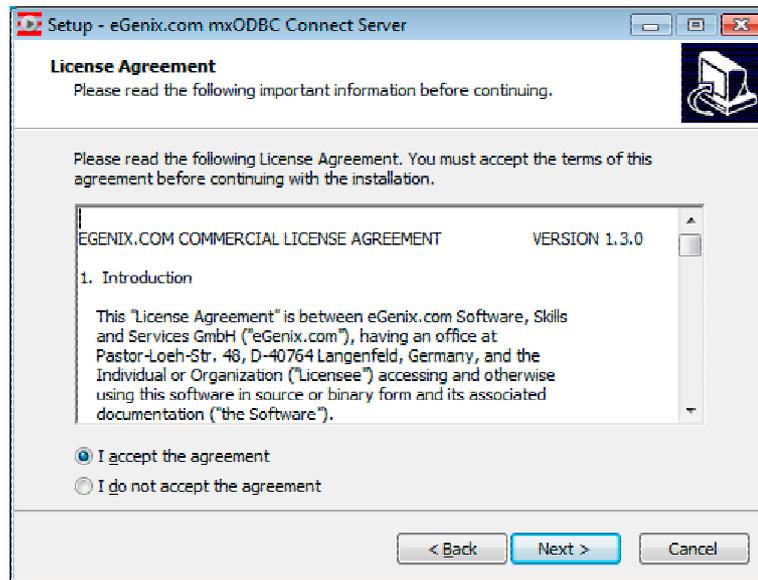
Next, you have to select the location where the server should be installed on your local disk drive. In most cases, you can simply accept the default value. If the folder does not exist, the installer will ask you whether it should create it.
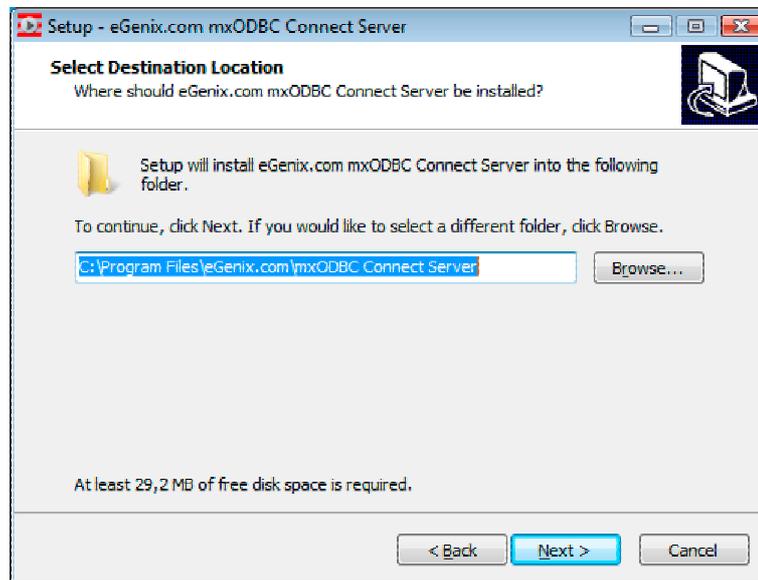
The server will also install a startup menu which provides access to the documentation, the uninstaller and a few utilities:
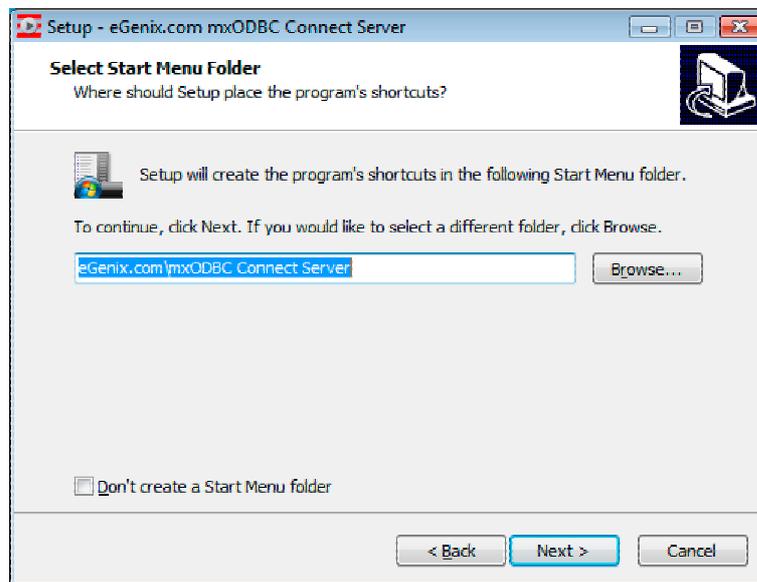
The installer is now ready to install the server. Please check the settings and then click on "Install".
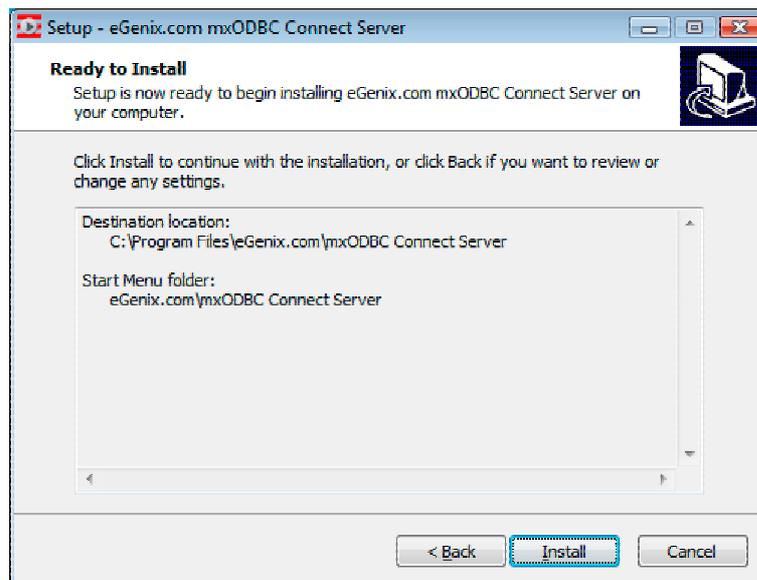
During the installation process, the installer will create example server and client SSL certificates that you can use to setup the server to accept SSL encrypted connections. See section 2.4 mxODBC Connect Server Configuration for more details.

The installer will also ask you for the mxodbc_connect_server_license.py file that you should have obtained from eGenix.com by email when signing

up for an evaluation or after purchase of an mxODBC Connect Server license.
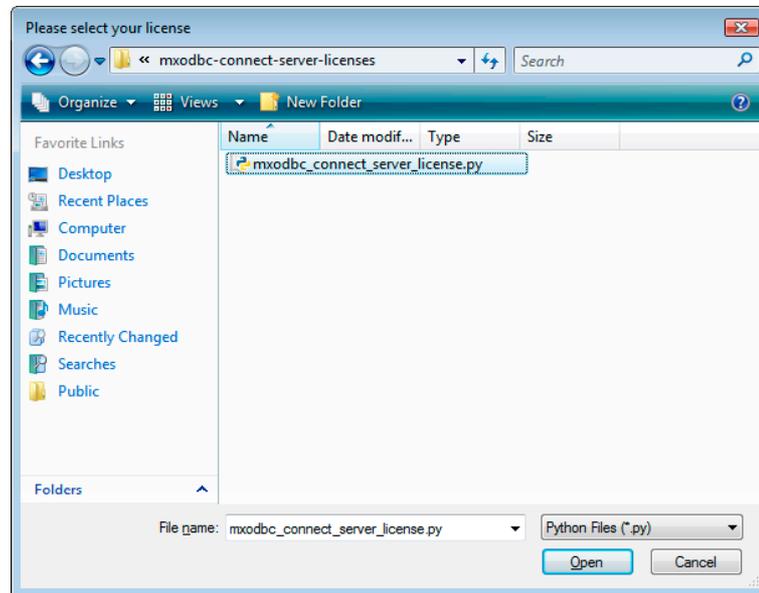
If you haven't yet downloaded and unzipped the license archive that was attached to the eGenix email, please do so now and point the installer to the location where you extracted the mxodbc_connect_server_license.py and mxodbc_connect_server_license.txt file for your installation.

Note that the installer will just ask for the mxodbc_connect_server_license.py file and expect the mxodbc_connect_server_license.txt to be in the same directory.

After installation of the server and license files, a read-me style notice will be displayed with a few helpful tips that you should read carefully:
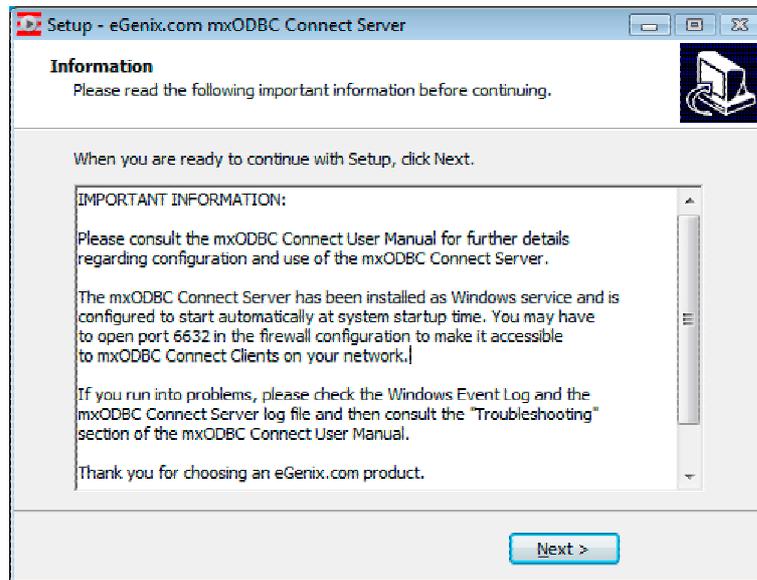
As last step, the installer will ask you whether it should automatically start the eGenix mxODBC Connect Server and the associated tray icon. If you choose not to start the server and/or tray icon application, you can always go back to the start menu to start them manually.
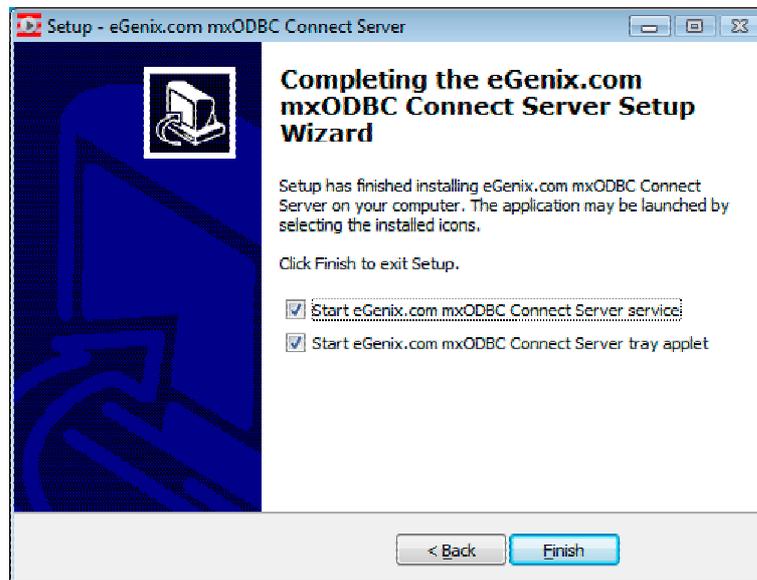
Click "Finish" to exit the installer.

Server Tray Icon

If you have enabled both checkboxes in the final dialog of the installation wizard, you should now see a small red icon ▣ in your system tray bar.

This icon indicates whether the eGenix.com mxODBC Connect Server is running (red arrow and icon) or not (grey cross and icon).

A right-click on the icon will open a menu that allows you to easily start, stop and restart the server. There are also shortcuts to open the configuration file, configure user access, check the server log file and open the documentation.

Configuring the Firewall

The installer registers a service named *mxODBC-Connect-Server*. This will listen on TCP port **localhost:6632** by default, i.e. the server will only accept connections from the local machine on port 6632.

TCP port 6632 is used for both plain and secure (SSL) connections. It's a IANA registered service port for eGenix mxODBC Connect with name *mxodbc-connect*.

> You may have to configure your firewall to allow connections on **port 6632** if you want to permit connections from a local subnet.

Edit the Configuration

After installation, you must **edit the configuration file** of the mxODBC Connect Server to fit your needs, e.g. have it listen for connections from the local subnet, and then restart the service in order for the changes to take effect. See section 2.4 mxODBC Connect Server Configuration for details on how this is done.

Controlling Automatic Startup of the Server

The mxODBC Connect Server service will be started automatically on each system startup by default. The startup type can be changed in the *Control Panel* / *Administrative Tools* / *Services* Windows panel or an similar configuration tool after installation.

Troubleshooting

If the mxODBC Connect Server service fails to start, please have a look at the server log file and consult section 7. Troubleshooting. The server log file is available via the tray icon menu entry *Show Log File*.

If you still have problems, please contact eGenix.com Support: *support@egenix.com*.

2.2.3 Uninstall

The Windows installer will automatically register the installed software with the standard Windows Software Setup tool.

To uninstall the server, run the Windows Software Setup tool and select the "*eGenix.com mxODBC Connect Server x.x.x*" entry for uninstallation.

This will stop and unregister the server service, then uninstall all files that can be safely removed from the system. It will not remove the configuration files, log and certificate files by default. You are asked whether to remove those as well at the end of the uninstallation process.

We suggest to backup your configuration directory before removing it.

2.2.4 Reinstallation or upgrading

You can reinstall or upgrade the mxODBC Connect Server by simply uninstalling it, without removing its configuration directory and then proceeding with the installation of the upgrade.

> **Note:**
> After upgrading, please check your server configuration file server-config.ini and compare it to the new default configuration file which will have been installed as server-config.ini.original.

It is also a good idea to review your changes of the server configuration against the new mxODBC Connect User Manual.

2.3 mxODBC Connect Server Installation on Unix

On Linux, mxODBC Connect Server is installed using a command line installer. The installer includes support for iODBC and unixODBC ODBC managers, one of which is usually preinstalled on Linux systems.

You can use the available GUI-configuration helpers for these ODBC managers to configure your ODBC data sources, which then become available to the mxODBC Connect Server and can be used by all applications connecting to the server using the mxODBC Connect Client.

2.3.1 Prerequisites

- You need *root* access to the target machine.

- Please make sure that you have a working unixODBC, iODBC or DataDirect ODBC manager installation prior to continuing with the installation. For details, please see the *mxODBC User Manual and Reference Guide*. In particular, the mxODBC Connect Server process needs to be able to access the ODBC manager configuration files, libraries and any ODBC drivers and driver specific files you may want to use with mxODBC Connect.

- You will need Unix ODBC drivers for all databases you wish to connect to. If you can't find the driver you are looking for have a look at section 8 *Hints & Links to other Resources*.

2.3.2 Procedure

Please download the binary distribution of mxODBC Connect Server for your version of the installed OS.

Extract the binary distribution (use the name of the file you downloaded):

```
tar -xf egenix-mxodbc-connect-server-x.x.x-linux-i686.tar.gz
```

Enter into the subdirectory created by tar and then execute the installer script:

```
./install
```

Then follow the instructions of the installer script in order to install the server.

If you run into a problem during the installation process, please consult section 7. Troubleshooting.

> **Note:**
> You have to uninstall any old version of the mxODBC Connect Server prior to installing a new version. The installer will ask you to do so and assist in uninstalling your old server.

Step-by-step Installation

The following is a transcript of a typical installation session:

```
$ ./install

===========================================================================
Welcome to the eGenix mxODBC Connect Server 1.0.0 installer/uninstaller !
---------------------------------------------------------------------------

The eGenix mxODBC Connect Server product requires that
you set up an account that can print be used by the server.

Please enter the account name. Both a group and user with
the given account name will be created, if they don't already
exist in the system.

If you have an existing installation of eGenix mxODBC Connect Server,
please enter the account name used by that installation.
---------------------------------------------------------------------------
Account name [mxodbc] :
```

The server is run under the given user account and group. Both are created by the installer automatically.

```
Where should the eGenix mxODBC Connect Server be installed on the system ?
---------------------------------------------------------------------------
[/opt/eGenix/mxODBC-Connect-Server] :

The directory /opt/eGenix/mxODBC-Connect-Server does not exist.
---------------------------------------------------------------------------
Create it ? (yes/no) [yes]
```

The default location of the server is under the /opt directory. You can change this directory to a different one, but care must be taken to make sure that the user account's home directory is set to the same directory.

```
Creating group 'mxodbc'...

Creating user account 'mxodbc'...

Please extract the license archive you received from eGenix and enter
the pathname of the directory containing your license files
(license.py and license.txt).
---------------------------------------------------------------------------
[/opt/eGenix/mxODBC-Connect-Server] :

*** Could not find the license.py file in that directory. Please retry.
```

The license files have to be unzipped in the newly created server directory. If the installer cannot find the files, it continues asking for a new directory until it succeeds.

```
---------------------------------------------------------------------------
[/opt/eGenix/mxODBC-Connect-Server] :
```

18

```
Installing application files...

Setting up file permissions...

Creating initial example certificates...
```

The installer will create a set of server and client certificates that can be used to setup SSL connections. You can replace these later on with your own certificates if needed.

```
The eGenix mxODBC Connect Server product comes with an init
script that can be used to automatically start the server
when the system starts up.

-------------------------------------------------------------------------
Install and enable the init script ? (yes/no) [yes]

Installing init.d script mxodbc-connect-server...
```

The init script provides a convenient way of starting/stopping the server. The installer will try to register the script with the system, but this may not always work due to the many ways of how Unix systems expect this to be done.

```
=========================================================================
eGenix mxODBC Connect Server 1.0.0 was successfully installed.

Please edit and update the configuration file:

  /opt/eGenix/mxODBC-Connect-Server/server-config.ini

and start the server using:

  /etc/init.d/mxodbc-connect-server start
```

You can now start the server for the first time and check the ~mxodbc/server.log file for successful startup.

Server User Account and Group

mxODBC Connect Server runs as it's own **mxodbc user** for security reasons and stores all of it's configuration and log files under the home directory of this user.

The user account and directories are created automatically during the installation process.

Configuring the Firewall

The installer registers a daemon named *mxodbc-connect-serve*. This will listen on TCP port **localhost:6632** by default, ie. the server will only accept connections from the local machine on port 6632.

TCP port 6632 is used for both plain and secure (SSL) connections. It's a IANA registered service port for eGenix mxODBC Connect with name *mxodbc-connect*.

You may have to configure your firewall to allow connections on this port if you want to permit connections from a local subnet.

Edit the Configuration

After installation, you must **edit the configuration file** of the mxODBC Connect Server to fit your needs, e.g. have it listen for connections from the local subnet, and then restart the service in order for the changes to take effect. See section 2.4 mxODBC Connect Server Configuration for details on how this is done.

Starting/Stopping the Server

The installer will ask you to start the server at the end of the installation.

This can easily be done using the provided *init.d* script (if you chose to install it):

```
/etc/init.d/mxodbc-connect start
/etc/init.d/mxodbc-connect restart
/etc/init.d/mxodbc-connect stop
```

or by running the server as *mxodbc* user directly:

```
sudo -u mxodbc ~/bin/mxodbc-connect-server start
sudo -u mxodbc ~/bin/mxodbc-connect-server restart
sudo -u mxodbc ~/bin/mxodbc-connect-server stop
```

Controlling Automatic Startup of the Server

The installer will attempt to register the *mxodbc-connect-server* daemon and add the appropriate *init.d* script to your system.

This may not always work due to the many different ways Unix derivatives implement the system startup process.

Please test the automatic server startup after reboot, prior to installing the server on a production machine.

Troubleshooting

If the mxODBC Connect Server service fails to start, please have a look at the server log file and consult section 7. Troubleshooting. The server log file is available in the home directory of the *mxodbc* user as server.log file.

If you still have problems, please contact eGenix.com Support: *support@egenix.com*.

2.3.3 Uninstallation

To uninstall the mxODBC Connect Server, run the `uninstall` script from your binary distribution or the server's application directory:

```
./uninstall
```

This will guide you through the uninstall process. The uninstaller will ask you whether you would like to keep the configuration files.

If you answer yes, only the product files that can be safely removed from the system will be uninstalled.

If you answer no, the complete installation will be removed - including any configuration files and/or certificates, the *mxodbc* user account and group.

> **Note:**
> The complete removal mode will also delete any customizations you have applied to the server configuration, so be sure to backup your configuration files and certificates before uninstalling, if you intend to reinstall the server in the future.

2.3.4 Reinstallation or upgrading

Running the `./install` script from the newly downloaded installer will reinstall or upgrade the server. The installer will automatically take care of uninstalling old components and replace them with updated versions.

> **Note:**
> After upgrading, please check your server configuration file server-config.ini and compare it to the new default configuration file which will have been installed as server-config.ini.original.

It is also a good idea to review your changes of the server configuration against the new mxODBC Connect User Manual.

2.4   mxODBC Connect Server Configuration

The server configuration INI file is named **server-config.ini** and located in the configuration directory of the mxODBC Connect Server. It's location depends on the operating system:

- **Windows:**
  Located in your documents and settings folder (may be called differently on non-English Windows versions), in the All Users\Application Data\eGenix.com\mxODBC Connect Server folder.

- **Linux:**
  Located in **~mxodbc**, i.e. the home directory of the *mxodbc* user that was created during the installation process.

Please make sure the server daemon has read access to all of its configuration files. It also needs write permission for its home directory in order to create and append to the log file. Note that the name and location of the log file can be configured.

The installer will configure the access rights for you. You only need to take special care when relocating the server installation or otherwise modifying its setup.

2.4.1   mxODBC Connect Configuration File Syntax

The mxODBC Connection configuration files use an INI-file like syntax:

```
global_option = 2

[Section1]
option_a = 1
option_b = abc.html
option_c = text with spaces

[Section2]
option_a = 2
option_b = 3
option_c = a string
```

The INI-file structure is the same for all supported platforms, both on the server and the client side.

You can use Unix or Windows line endings and the forward slash ("/") as path separator on both systems, but it's recommended to use the backslash ("\") on Windows.

The INI files are parsed using the following rules:

- Entries in square brackets indicate new subsections.

- Global variables may be set prior to starting any subsection.

- Empty lines and lines starting with '#' or ';' (comments) are ignored.

- Indentation is not necessary. Lines can start at any column.

- Entries may span multiple lines by using '\' continuations at the line ends. The lines are stripped of any white space before removing the trailing '\' and concatenating them. Comment lines are removed as well.

  Example:

  ```
  [Continuation]
  a = first line\
      second line
  ```

Some additional notes regarding the INI-file format used by mxODBC Connect:

- Comments may be used in the INI-file, but only on separate lines, i.e. a comment after a value is not permitted and will likely cause problems when parsing the option value.

- All pathnames in the configuration file are relative to the directory of the configuration file. You can use absolute pathnames to point any file in the file system.

2.4.2 mxODBC Connect Server Configuration File

The configuration file uses a standard INI-file format (see section 2.4.1 on page 22) and has the following sections and options with their default values (some are OS dependant).

Most settings have default values, so you only need to provide those settings which you intend to change from their default.

All file names defined in the server configuration file are interpreted as relative to the server configuration file. If you intend to change these file names to locations outside the normal server installation or configuration

directory, please make sure that the server has permission to access these files and/or directories.

[Connection_Name]

These named sections each define a network connection to be opened and managed by the server.

You can add more sections with different names to define multiple connections of your server. The only requirement is that the section names contain the term "Connection" or "connection".

Please use distinctive section names such as *Connection_Local*, *RemoteConnection* or *CompanyVPNConnection* to prevent future collision with predefined section names.

All connection sections have the following common attributes:

`interface = 127.0.0.1`

> IP address of the network interface to listen on.
>
> 127.0.0.1 will only allow connections from the local host.

`netmask = 255.0.0.0`

> Netmask of the interface.
>
> You should adjust this setting to the layout of the subnet. The server will only allow connections from the subnet defined by the interface IP address and netmask.

> **Note:**
> IP based access control is not considered as a real security feature. You need an SSL connection, precise rules for database access and difficult to guess passwords to secure your database server.

`port = 6632`

> Port number to listen on.
>
> Default port number is 6632 (IANA name *mxodbc-connect*) which is used for both plain and secure (SSL) connections and is a IANA registered port for eGenix mxODBC Connect.
>
> You are free to use any free port number, as long as you follow the convention that unprivileged users and non-standardized services need to use port numbers above 1024. However, any change you make on the server side will also have to be reflected on the client side.

> **Note:**
> Using obscure port numbers will not increase your security, since a simple port scanner utility can reveal your port number.

Advanced Options

You typically do not need to modify the following options. Not specifying them will have the server use the given default values.

`allow_reuse_address = 1`

> Enables the SO_REUSEADDR socket option.
>
> This socket option tells the kernel to use the port even if it is currently busy in the TIME_WAIT state. If the port is busy, but with another state, you will still get an "address already in use" error.
>
> This option is useful when restarting the server daemon.

`keepalive = 1`

> Enables SO_KEEPALIVE socket option, which ensures that client connections will not be dropped due to long inactivity.
>
> It is recommended to keep this option enabled.

`request_queue_size = 10`

> The TCP request queue size.
>
> Maximum number of client connections that can wait to be accepted. Increase only if you expect a large number of connection attempts per second.

`socket_timeout = None`

> TCP socket timeout in seconds or None for disabling connection timeout.
>
> This is the length of inactivity period after the TCP connection should be dropped.
>
> You should not need to modify this option. Use `[Activity]` `max_waiting_time` instead.

Options for SSL Encrypted Connections

The following options below are only relevant, if SSL is to be used on the connection.

`allow_ssl = 0`

Setting `allow_ssl` to a non-zero value enables the secure socket layer (SSL) support on this connection.

You can use SSL to encrypt all communication and also to authenticate clients via certificate verification.

Note that the server can handle both plain text and SSL connections on the same port, if `allow_ssl` is set to a non-zero value. If you want to disable plain text connections, set `require_ssl` to a non-zero value instead.

Default is to only accept plain text connections (`allow_ssl = 0`).

`require_ssl = 0`

Setting `require_ssl` to a non-zero value disables plain text connections on this connection. The client has to start a SSL connection if it wants to communicate with the server.

If the `require_ssl` setting is enabled, `allow_ssl` is enabled implicitly, i.e. set to a true value.

Default is to accept both plain text and SSL encrypted connections, if `allow_ssl` is enabled.

`server_private_key_file = server.pkey`

Name of the server's SSL private key file.

This option is required for all SSL connections. The installer provides a default, self-signed key pair. You can replace it with your own PEM-encoded private key file .

`server_certificate_file = server.cert`

Name of the server's SSL certificate file.

This option is required for all SSL connections. The installer provides a default, self-signed key pair. You can replace it with your own PEM-encoded certificate file .

`client_certificate_file = has no default value`

Name of the file that contains concatenated certificates for client certificate verification.

`client_certificate_dir = has no default value`

Name of a directory that contains files with single certificates for client certificate verification.

`client_certificate_digest = has no default value`

Space separated list of SHA1 digest values of accepted client certificates.

It's recommended to use `client_certificate_digest_file` if you have many digest values to prevent cluttering up the configuration file and allow sharing of digest list between connections.

`client_certificate_digest_file = has no default value`

Name of a file that contains SHA1 certificate digests for client certificate verification.

The file can contain more than one digest value, one per line.

Client Certificate Access Rules

A client may connect, only if at least one of the above client certificate verification rules matches the certificate presented by the client.

If none of the verification options are defined then all clients are accepted, regardless the content of their client certificates. The server log file lists the SHA1 digest values of all accepted certificates on each server start. It also logs the SHA1 digests of all client certificates it accepts, if one of the verification options is enabled.

Connection Default Configuration

To simplify adjusting the mxODBC connection default format/parameter values without having to change the applications using the mxODBC Connect Client, the mxODBC Connect Server provides the following configuration settings.

These connection settings are applied to the mxODBC connection directly on the server side and right after a connection is opened.

`connection_cursottype = None`

Defines the `connection.cursortype` to set after opening the connection on the server side.

The cursor type defines how the database handles result sets opened with the cursor. For more information, please check the documentation of your ODBC driver and the *mxODBC manual*.

The server configuration provides a special object called `SQL` to make configuration of this setting easier.

Possible values are:

- `SQL.CURSOR_FORWARD_ONLY` - cursor can only scroll/move forward; this is the fastest cursor type variant, but does not make any guarantees with respect to result set membership or order

- `SQL.CURSOR_KEYSET_DRIVEN` - result set keys are stored, so the result set membership and order does not change; changes to the result set row values are possible

- `SQL.CURSOR_DYNAMIC` - result sets are dynamically updated with changes from other users; membership and order can change while processing the result set

- `SQL.CURSOR_STATIC` - result set does not change after opening the cursor; membership and order or the rows are maintained while processing the result set

- `None` - the default value for `.cursortype` as identified by mxODBC is used. Please see the *mxODBC manual* and your ODBC driver documentation for details on how mxODBC sets up the cursors per default. This is the default value for the `connection_cursortype` setting.

Please note that not all databases support all of the above cursor types. Only the `SQL.CURSOR_FORWARD_ONLY` type is supported by all databases.

[Authentication]

The mxODBC Connect Server can be protected against unauthorized access using different authentication mechanisms. This section configures how authentication is handled by the server.

Note that these authentication checks are not very secure. It is generally better to use SSL connections only and implement access control via client certificate checking than relying just on authentication using a username and a password.

`auth_mode = none`

Authentication mode to use.

Possible values:

- `none` - no authentication

- `file` - password file based authentication.

`password_file = authorized-users.txt`

If `auth_mode` is set to `'file'`, `password_file` must point to a text file defining the users that are allowed to access the mxODBC Connect Server.

The file format of the password file uses one line entries of the form "user: sha-256-hex-digest$salt$version" for each user. Empty lines and

28

lines starting with '#' or ';' are ignored. The hash values must be generated using the password-tool included with the server distribution.

Please see section 2.4.5 Configuring User Authentication for details.

The file should only be readable by the mxODBC Connect Server daemon.

`login_salt = <internal default>`

In order to provide some extra protection when sending the login request over the network, client and server can be configured to add a salt string to the hashed login credentials.

Only set this, if you want to override the internal default or need to separate multiple mxODBC Connect installations from each other.

The salt string should have at least 16 bytes and should not contain spaces. If given, the server setting for this variable must match those of the clients that want to connect to the server. The `login_salt` can be thought of as shared secret.

[Session]

This section controls the details of the communication between the server and clients. Each client will normally open one session to the server. A session can host multiple physical database connections.

`max_sessions = 400`

Maximum number of concurrent sessions allowed by the server.

This parameter is intended to prevent Denial of Service (DOS) attacks on the server. You can also use it for debugging purposes or to reduce the load on the server. Clients will get a connection error in case the server has reached the maximum number of concurrent sessions.

Note that this is not the same as the number of concurrent database connections. Those are limited by the server license you have installed.

`enable_compression = 1`

Network communication compression.

Setting this variable to 1 will enable compression of TCP packets sent by the server, setting the variable to 0 causes compression to be disabled.

Compression is enabled per default, in order to reduce network traffic and enhance roundtrip times.

On very fast networks or local connections you may want to disable compression for enhanced performance. We have found that even on

Gigabit Ethernet networks, enabling compression does provide a performance increase.

`compression_ratio = 2`

Compression ratio to use for network communication compression.

Valid values are 1 (least compression, fast) **-** 9 (best compression, slow).

The default value of 2 is a good compromise for fast networks.

You may want to experiment with the setting to tune it for best performance on your network.

In some setups, e.g. fast server and slow clients, it may be wise to use different compression ratios for clients and servers. The server setting affects packets sent from the server to the client, whereas the client setting affects packets sent from the client to the server.

`max_chunk_length = 64000`

Maximum chunk length for TCP read/write operations.

You will normally not need to change this value.

`receive_timeout = 10`

Timeout for one TCP receive operation in seconds.

You will normally not need to change this value.

`send_timeout = 10`

Timeout for one TCP send operation in seconds.

You will normally not need to change this value.

This value should allow fairly fast transfers. You normally don't need to modify it, unless the server or client platform have specific requirements with respect to TCP packet sizes.

[Unix]

This section is only used on Unix systems, such as Linux and FreeBSD.

`pid_file = server.pid`

Name of the PID file to store the server's PID.

[Windows]

This section is only used on Windows and currently does not have any options.

[Activity]

This section defines settings which affect the way it handles timeouts and administrative tasks.

`check_interval = 2`

> The time period in seconds of checking server threads for finished threads and errors. This is required to clean up internal data structures.

`max_request_execution_time = 86400`

> Maximum time allowed to execute a client request in seconds. This should be twice the time required to complete your longest SQL query. It is used to detect timeout conditions and needed to free resources.

`max_session_reconnect_time = 60`

> Maximum amount of time in seconds a working thread in the server will wait for a session reconnect after a communication failure.
>
> This is intended to allow intermittent communication failures not to cause an immediate disruption of the session based communication between client and server, while at the same time, preventing the server from holding on to resources such as database connection for longer periods of time after a communication failure.

`max_waiting_time = 86400`

> Maximum amount of time in seconds a working thread in the server waits for a command from a client. After this time the session will be dropped and the client must reconnect.
>
> The client can prevent the timeout by executing dummy like operations, such as a query that doesn't return any values (e.g. `"SELECT 0 WHERE 1=0"`). The client could then also catch any connection related exceptions and reconnect as necessary.

[Logging]

This section defines the details of logging output.

`log_level = mx.Log.SYSTEM_IMPORTANT`

> Log level. Please see mxLog for details.
>
> Commonly used log levels (ordered from only logging serious problems to logging everything):
>
> - `mx.Log.SYSTEM_ERROR` - log error messages (but no less important ones)

- `mx.Log.SYSTEM_IMPORTANT` - log important messages (but no less important ones)

- `mx.Log.SYSTEM_WARNING` - log warning messages (but no less important ones)

- `mx.Log.SYSTEM_MESSAGE` - log operational messages (but no less important ones)

- `mx.Log.SYSTEM_INFO` - log informational messages (but no less important ones)

- `mx.Log.SYSTEM_ANY` - log all messages

The server configuration system provides access to the `mx.Log` module for the purpose of defining these values.

`log_file = server.log`

Name of the log file to use. Please ensure, that the server can create the log file and append to existing one. Failing to do so will kill the server with access denied exception.

2.4.3 Server Connection Setup

In order to accept connections from the network, you will have to customize the server configuration to your needs. This section explains how connections are setup.

The mxODBC Connect Server supports listening on multiple ports and interfaces. Each connection configuration needs to be placed into its own section of the configuration file. The server detects the connection sections by looking at the section title. All sections that start with "Connection" are interpreted as connection configuration sections.

Basic configuration

This is a basic configuration which listens on subnet 129.168.0.12/24 for plain text (unencrypted) connections:

```
[Connection_Office]
    interface = 192.168.0.12
    netmask = 255.255.255.0
```

This connection will not accept SSL-encrypted connections and also reject any connections from other subnets.

Adding SSL support is easy

The server installation will create two certificate files for you: server.pkey (the private key file) and server.cert (the public key certificate file). You can use these generated files as initial setup and later on replace them with your own public key certificate files, if you wish.

Since the above two file names are used per default, enabling SSL connections on the server port is done by simply adding `allow_ssl = 1` to the connection configuration:

```
[Connection_Office]
    interface = 192.168.0.12
    netmask = 255.255.255.0
    allow_ssl = 1
```

If you are using custom certificates, you can also point the server to those files:

```
[Connection_Office]
    interface = 192.168.0.12
    netmask = 255.255.255.0
    allow_ssl = 1
    server_private_key_file = my-private-key-file.pkey
    server_certificate_file = my-public-key-file.cert
```

> Note that using `allow_ssl = 1` will not force clients to connect using SSL. Plain-text connections are still possible as well. Please see the next section on how to disable plain-text connections altogether.

Even more secure: SSL-only connections

The communication can also be restricted to SSL-only. This effectively disables plain text connections on the server port. All you have to do, is add the `require_ssl = 1` line to the configuration:

```
[Connection_Office]
    interface = 192.168.0.12
    netmask = 255.255.255.0
    require_ssl = 1
```

Listening on more than one port

If you have special setup requirements where you want to use more than one port, e.g. one for plain text connections and one for SSL-only connections, you can define more than one connection section in the configuration file:

```
[Connection_Office]
    interface = 192.168.0.12
    netmask = 255.255.255.0
    port = 6632

[Connection_Office_SSL]
    interface = 192.168.0.12
    netmask = 255.255.255.0
```

```
    port = 6633
    require_ssl = 1
```

The above setup emulates the setup which was used by mxODBC Connect 1.0, where two ports were used for the communication, one for plain-text, the other for SSL connections. With mxODBC Connect 2.0 and later, this is no longer necessary, since the server can now switch between plain-text and SSL as necessary.

2.4.4 Configuring Certificate Based Authentication

The mxODBC Connect Server can perform certificate based authentication checks when a client connects to it via SSL. For this to work, the connection needs to be configured to require SSL, e.g.

```
[Connection_SSL]
    interface = 192.168.0.12
    netmask = 255.255.255.0
    require_ssl = 1
```

The server provides these ways of securing SSL connections:

- providing a file which contains all allowed client certificates

- placing the allowed client certificate files into a directory

- providing a list of allowed client certificate SHA1 hex digests in the server configuration file

- providing a list of allowed client certificate SHA1 hex digests in a separate file

Important: All options are read and processed at server startup time, so any change will only take affect after a server restart.

Using a file with client certificates

With this option, the incoming client certificates are checked against a file which contains the allowed client certificates concatenated in PEM format[2].

To enable this client certificate check, please add the client certificates to a file on the server and then add the path to this file to the configuration file, e.g.

```
# Name of the file that contains concatenated certificates for
# client certificate verification.
```

_____

[2] PEM format is a special text file format, which can easily be edited using a text editor.

```
client_certificate_file = allowed_client_certificate.cert
```

The path may be given relative to the server configuration file's directory.

Using a directory with client certificates

With this option, the incoming client certificates are checked against a directory listing the allowed client certificates in PEM file format. The files have to use the extension ".cert" to be included in the search.

To enable this client certificate check, please add the client certificates to a directory on the server and then add the path to this directory to the configuration file, e.g.

```
# Name of a directory that contains files with single certificates
# for client certificate verification
client_certificate_dir = allowed_client_certificates/
```

The directory may be given relative to the server configuration file's directory.

Using a list of SHA1 hex digests in the configuration file

With this option, the incoming client certificates are checked against a list of allowed SHA1 certificate digests.

On Unix and provided you have OpenSSL installed, you can determine the SHA1 certificate digest by running the following command against the client certificate:

```
> openssl x509 -fingerprint -in client.cert
SHA1 Fingerprint=
88:EA:FA:AD:1C:CB:2D:34:9B:07:6D:2B:5C:0C:22:23:9F:F5:03:32
```

On Windows, you can rename the client.cert file to client.crt and then use the Windows Explorer certificate helper to open the certificate. The details section will show the SHA1 hex digest.

To enable this client certificate check, please add the SHA1 digests in hex format to the configuration file as space separated entry, e.g.

```
# Space separated list of SHA1 digest values of accepted client
# certificates
client_certificate_digest = \
    0C5BD019D9A5C2D4279CC3E4E340E17F \
    0C5BD019D9A5C2D4279cc3E4e340e180 \
    0C5BD019D9A5C2D4279CC3E4E340E181
```

You can use the backslash (" \") at the end of a line to split the setting across multiple lines. Please remove any embedded spaces, colons or dashes from the hex digests before adding them.

Using a file with SHA1 digests

With this option, the incoming client certificates are also checked against a list of allowed SHA1 certificate digests. In this case, the digest values are read from a file.

Please see the above section for how to extract the SHA1 digest from the client certificates.

To enable this client certificate check, please add the SHA1 digests in hex format to a file and then add the path to this file to the configuration file, e.g.

```
# Name of a file that contains SHA1 certificate digests for client
# certificate verification
client_certificate_digest_file = allowed_clients.sha1
```

The file allowed_clients.sha1 must contain one SHA1 hex digest per line. Comment lines starting with '#' and empty lines are allowed, e.g.

```
# SHA1 hex digests of allowed certificates:
88:EA:FA:AD:1C:CB:2D:34:9B:07:6D:2B:5C:0C:22:23:9F:F5:03:32
B7:63:C4:85:E8:2A:F2:AD:C6:B6:1D:01:0D:AE:FE:D0:9B:46:B3:80
```

The digest lines may use colons, dashes or spaces as additional separators.

A malformed SHA1 digest entry will cause the server fail at startup with a notice to the log file.

2.4.5 Configuring User Authentication

The mxODBC Connect Server provides a user authentication mechanism to protect the server itself (not only the database) from unauthorized access.

> **User authentication is disabled by default** to make a first time configuration easier, but should be setup once the basic client-server communication has been configured and found working, unless you are using the more secure certificate based authentication.

To enable user authentication, edit the server-config.ini file and set the auth_mode setting in the [Authentication] section to 'file' (without the quotes).

Authentication Protocol

The authentication protocol implemented by mxODBC Connect follows a similar scheme as the HTTP Basic Authentication protocol and provides a comparable level of security.

It is usually better to always use SSL encrypted connections, to prevent someone from stealing database passwords, the session cookie of a logged in session or applying a replay attack to get access to the mxODBC Connect Server.

Password File authorized-users.txt

You can then either create a password file based on the example file authorized-users-example.txt shipped with the server or use the **password-tool** to create and administer the file.

When creating the file, it is a good idea to make sure that the file can only be read by the server daemon or service. It is also possible to change the default name authorized-users.txt of the password file by adjusting the `password_file` entry of the `[Authentication]` section.

The file format of the password file is similar to that of a web server password file.

User entries are of the form "`username: hash-value$salt$version`", with one line per entry. Empty lines, lines starting with '#' or ';' are ignored, so that you can add comments as necessary.

Example:

```
test1: a84fcb9 … b65c9a5e0c2d60e6f97efe2337b4924d2$7465737431$2.1
test2: c562370 … 3561c0102dbd19f87143576d1215b4aa2$7465737432$2.1
```

The salted SHA-256 hash values can easily be created using the password-tool command-line application for editing the file, since this provides all the necessary encoding of the password. As of mxODBC Connect 2.1, the authorized-users.txt file entries should no longer be created using external tools.

> Please note that even though we switched from simple MD5 hashes used in mxODBC Connect 2.0 and earlier to salted SHA256 hashes for better security, these password files are not as secure as OS level password files using more sophisticated algorithms. They just provide an additional level of security when used together with certificate based authentication.

Using the password-tool

The *password-tool* command-line application is available as ~/bin/password-tool in the Unix installation of the server and as password-tool.exe in the Windows installation. Both provide a command-line option-based interface and an interactive shell-like interface to edit the password file.

Command-line Options of the password-tool

The password-tool application prints out a list of options when started with
`-h` option:

```
------------------------------------------------------------------
eGenix mxODBC Connect Server - Password Tool
------------------------------------------------------------------

Synopsis:
 password-tool [options]

Options and default settings:
  -f arg          password file (authorized-users.txt)
  --file arg      password file (authorized-users.txt)
  --add arg       add user arg
  --update arg    update user arg
  --delete arg    delete user arg
  --list          list all entries
  -p arg          password
  --password arg  password
  -v              generate verbose output
  --verbose       generate verbose output
  -h              show this help text
  --help          show this help text
  --debug         enable debugging
  --copyright     show copyright
  --examples      show examples of usage

Note: When started without options, the script goes into
interactive mode.
```

Options explained in more detail:

`--file arg`

Edit the password file `arg`.

The default is to use the file specified in the server configuration as
password file.

`--add arg`

Add a new user `arg` to the password file.

`--update arg`

Update the user `arg`'s password file entry, ie. set a new password.

`--delete arg`

Delete user `arg` from the password file.

`--list`

List all entries currently found in the password file.

`--password arg`

> Provide the password to use for any subsequent action on the command line.

> If this option is not used, the script will read the password from the terminal or `stdin` (without displaying it).

Interactive Mode of the password-tool

When called without any options, the password-tool goes into an interactive mode which allows editing the password file using a set of basic commands.

On Windows, this can also be done by clicking on the product's "*Configure User Access*" start menu entry or from the tray icon menu.

After start-up, the password-tool shows a dialog and asks for action command input:

```
-----------------------------------------------------------------
eGenix mxODBC Connect Server - Password Tool
-----------------------------------------------------------------

Possible actions:

    add     - add a new user
    update  - update an exiting user
    delete  - delete an existing user
    list    - list all users
    quit    - quit the application

>>>
```

You enter the action commands at the ">>>" prompt, followed by return. Only the first character of the actions has to be entered.

The various actions will then ask for more input as necessary.

To exit the password-tool, enter "q", followed by return.

> **Note:**
> In interactive mode, the password-tool will always edit the password file configured in the server-config.ini file, usually authorized-users.txt.

2.5 ODBC Driver Configuration Hints

The typical installation of a mxODBC Connect will have the server part installed directly on the database server and the client parts on the machines running the application.

2.5.1 Setting up the optimal communication technique

In order to benefit from the locality of having the mxODBC Connect Server installation running directly on the database, you have to make sure that the ODBC data sources configured on the database server use the best available communication protocol for connecting to a local database server.

Choosing a TCP/IP connection type for the ODBC data sources will not give you the best performance.

If possible, you should select communication options such as *Shared Memory*, *Named Pipes*, *Domain Sockets*, or similar communication methods that allow fast and direct communication between the ODBC driver and the database kernel.

For MS SQL Server 2000 this option would be *Named Pipes*. MS SQL Server 2005 and later also support the more efficient *Shared Memory* communication method.[3]

Please refer to your database documentation on how to setup the ODBC driver and database for using the optimal communication technique for local connections.

---

[3] MS SQL Server 2005 and later use the *SQL Server Native Client* as ODBC driver. The communication protocols for this driver are defined in the *SQL Server Configuration Manager*.

2.5.2 Disabling options that are not needed for local connections

Also make sure that you have additional features such as **connection encryption** switched off for ODBC data sources that you intend to use with mxODBC Connect Server. Since the communication never leaves the server, encrypting it would only cause a performance hit and not result in better security.

3. mxODBC Connect Client Installation

The mxODBC Connect product consists of a stand-alone server component and client packages for various platforms. The installers for both components are distributed separately.

You only need to integrate the client side package in your application for ODBC functionality over the network. With mxODBC Connect it is no longer necessary to have an ODBC driver installed on the machine where you run your Python-based applications.

The mxODBC Connect Client package is distributed as an add-on for the eGenix.com mx Base Distribution (`egenix-mx-base`). Please visit *http://www.egenix.com/products/python/mxBase/* to download the latest version of the eGenix.com mx Base Distribution for your platform and Python version.

If you also want to benefit from encrypted connections between the mxODBC Connect Client and Server, then you additionally need the Python standard library module `ssl` installed, which is available in Python 2.6 and later, or the eGenix.com pyOpenSSL Distribution (`egenix-pyopenssl`). Please visit *http://www.egenix.com/products/python/pyOpenSSL/* to download the latest version of the eGenix.com pyOpenSSL Distribution.

> **IMPORTANT NOTE:**
>
> Before installing the `egenix-mxodbc-connect-client` package, you will have to install the `egenix-mx-base` distribution which contains packages needed by mxODBC Connect.
>
> Even though both distributions use the same installation procedure, please refer to the `egenix-mx-base` installation instructions on how to install that package.

3.1 Upgrading mxODBC Connect Client

3.1.1 Upgrading from 2.0 to 2.1

mxODBC 3.3 API

mxODBC Connect Server uses the new mxODBC 3.3 version on the server side, which provides better compatibility with current ODBC drivers and includes a number of important new features, most notably the addition of support for in/out and output parameters in SQL commands and stored procedures and the ability to specify alternative row constructors in a very efficient way.

Please see the *mxODBC User Manual and Reference Guide* for details on the mxODBC 3.3 API.

mxODBC Connect Client supports all features of the mxODBC 3.3 API, with the exception of a few details that are outlined in section 6. Differences between mxODBC and mxODBC Connect.

What follows is a quick overview of the changes and enhancements which are visible on the client side of mxODBC Connect. For server side enhancements, please have a look at section 2.1 Upgrading mxODBC Connect Server.

Stored Procedures

- mxODBC Connect now has **full support for input, output and input/output parameters** in **stored procedures** and **stored functions**, allowing easy integration with existing databases systems.

43

User Customizable Row Objects

- **Added support for user customizable row objects** by adding **cursor/connection .rowfactory and .row constructor attributes**. When set, these are used to wrap the normal row tuples returned by the .fetch*() methods into dynamically created row objects.

- **Added new RowFactory classes** to support cursor.rowfactory and cursor.row. These allow dynamically creating row classes that provide sequence as well as **mapping and attribute access to row fields** - similar to what namedtuples implement, but specific to result sets.

Fast Cursor Types

- **Switched to forward-only cursor types for all database backends**, since this provides a much better performance for MS SQL Server and IBM DB2 drivers.

- Added a **new .cursortype attribute** to allow **adjusting and inspecting the ODBC cursor type** to be used for an mxODBC Connect cursor object. Default is to use forward-only cursors, but mxODBC Connect also supports several other useful cursor types such as static cursors with full support for result set scrolling.

More new Features

- **Enhanced cursor.prepare()** to allow querying cursor.description right after the prepare step and not only after calling a cursor.execute*() method.

- **Added iterator/generator support to .executemany().** The parameters list can now be an iterator/generator, if needed.

- **Added new connection.dbapi** property to easily access module level symbols from the connection object.

- **Timestamp seconds fraction resolution** is now determined from the scale of a datetime/timestamp SQL column, using the connection.timestampresolution as lower bound, when using SQL type binding. In Python type binding, the connection.timestampresolution determines the scale with which a variable is bound. This allows for greater flexibility when dealing with database backends that don't provide full nano-second second resolution, such as e.g. MS SQL Server.

- mxODBC Connect now accepts **Unicode string values for date/time/datetime/timestamp column types** in SQL type binding mode.

Previous versions already did in Python type binding mode.

- **mxODBC Connect now uses unicode(obj, encoding) semantics** when binding Python objects to SQLWCHAR database parameters. Additionally, it ignores the encoding in case obj is a number, to avoid conversion errors.

- **Added new cursor.encoding attribute**. This gets its default values from the connection the cursor was created on.

- **Added cursor.bindmethod** which inherits from connection.bindmethod when creating the cursor. This allows adjusting the variable bind method on a per-cursor basis, rather than only on a per connection basis as in previous mxODBC Connect versions.

mxODBC Connect API Enhancements

- The **SQL lookup object is now cached on the client side** to avoid frequent roundtrips when using symbols which are needed for stored procedures with input/output parameters.

- The SQL lookup object now supports **ODBC 3.8 symbols** and values, including driver specific symbols used by the MS SQL Server Native Client and IBM DB2 ODBC drivers.

- **Improved the server side object management** to simplify client side garbage collection considerations. Even though we still encourage using explicit garbage collection of cursors, connections and server sessions on the client side, mxODBC Connect Server will now handle most situations even without these explicit calls.

Asynchronous Processing

- **Tested with the latest gevent and greenlet packages.** mxODBC Connect Client will happily work together with the asynchronous libraries gevent. All it takes is a single configuration entry in the client side config file.

Security Enhancements

- **Enhanced user authentication** to use salted SHA-256 hashed passwords when transferring login data from the client to the server. This provides better protection when using plain text client server setups.

3.1.2 Upgrading from 2.0.x to 2.0.4

New connection_cursortype server configuration parameter

In version 2.0.4 of the mxODBC Connect Server, we have added a new configuration setting to the [Connection] sections called `connection_cursortype`.

This allows you to pre-configure the new mxODBC `connection.cursortype` to a fixed value without having to change the client side application.

3.1.3 Upgrading from 2.0.x to 2.0.3

New .cursortype Attribute

We found a major performance problem with the static ODBC cursors used in mxODBC Connect 2.0.x. To address this, a new connection and cursor attribute `.cursortype` was backported from mxODBC Connect 2.1 to the 2.0.3 release.

Enhance MS SQL Server and IBM DB2 Fetch Performance

With this new attribute you can adjust the used ODBC cursor type easily. Specifically for Microsoft SQL Server and IBM DB2 using forward-only cursors instead of the default static cursors is strongly advised - unless you have a need for static cursors. Please see the *mxODBC User Manual and Reference Guide* for details on the various cursor types.

Here's an example of how to change your applications to benefit from the enhanced performance using forward-only cursors:

```
# Connect to the remote database
from mx.ODBCConnect.Client import ServerSession
session = ServerSession(...)
ODBC = session.open()
connection = ODBC.DriverConnect(...)

# Use the faster forward-only cursors
connection.cursortype = ODBC.SQL.CURSOR_FORWARD_ONLY

# Cursors created on this connection will then default to forward
# only cursors, instead of the mxODBC 3.2 default for SQL Server
# of using static cursors
cursor = connection.cursor()
```

By simply setting the `connection.cursortype` to forward-only cursors, all subsequently created cursors will use this new faster setting (see the highlighted lines in the above example.

3.1.4 Upgrading from 1.0 to 2.0

Network Related Changes

For the version 2.0 of the server, we have registered the port 6632 used by mxODBC Connect with IANA (as *mxodbc-connect* service).

Since assigned ports are a rare resource, port 6633 is no longer used by the server per default. However, you can still configure the server to use this port, if needed.

Port 6632 can now be used for both SSL and plain-text communication. It is even possible to have a mixed setup where some clients use plain-text and others use SSL communication over that port.

Configuration File Changes

Unlike version 1.0, version 2.0 is now using a single port to implement SSL and plain-text communication. As a result, the configuration setting `using_ssl` no longer switches the default port from 6632 to 6633.

> **If you have not changed your server configuration** to only use and listen on the single port 6632, you will have to explicitly add the port 6633 definition in the client configuration:

```
[Connection_RemoteServer]
host = database.example.net
using_ssl = 1
port = 6633
```

Please see section 2.1 Upgrading mxODBC Connect Server for details on how to update the server configuration, which allows avoiding such client side configuration changes.

mxODBC Feature Changes

mxODBC Connect Server uses the new mxODBC 3.2 version on the server side, which provides better compatibility with current ODBC drivers and also includes a number of new features compared to the older mxODBC 3.0 version included in mxODBC Connect Server 1.0.

Please see the *mxODBC User Manual and Reference Guide* for details on the mxODBC API changes.

mxODBC Connect Client supports all features of the mxODBC 3.2 API, with the exception of a few details that are outlined in section 6. Differences between mxODBC and mxODBC Connect.

> Note that unlike the mxODBC 3.2 stand-alone version, mxODBC Connect Client is compatible with *gevent*. See 5.3 gevent Support for details.

3.2 mxODBC Connect Client Installation on Windows

The mxODBC Connect Client is a regular Python package. It allows to connect your mxODBC compatible application to an ODBC compatible database over a network.

In order to connect to a database you need to run a properly configured mxODBC Connect Server on the machine with the target ODBC driver, usually the machine running the target database itself.

3.2.1 Prerequisites

- Python 2.5 or later installed. See *http://www.python.org* for details and download instructions.

- eGenix's mx Base extension installed. See *http://www.egenix.com/products/python/mxBase/* for details and download instructions.

- For SSL support (optional), you should either have eGenix's pyOpenSSL Distribution installed or use the Python standard lib module ssl. See *http://www.egenix.com/products/python/pyOpenSSL/* for details for details and download instructions of eGenix's pyOpenSSL distribution.

Note that using SSL encrypted communication is not allowed in all countries. Please check with your local authorities whether you are permitted to use encryption.

The client has been tested with the official Python 2.5, 2.6 and 2.7 installers. Python 2.4 and below are not supported.

3.2.2 Procedure

> **Note:**
> You may need administrative privileges on Windows XP/2003 and later to successfully complete the installation or un-installation process.

Please uninstall any existing version of mxODBC Connect Client if you have one installed (see section 3.2.3 for details).

Please download the Windows installer from *eGenix.com* that matches your Python version. Double click on the executable you downloaded to begin the installation process. Depending on the Windows version, you may have to click through a security dialog to proceed. Then follow the instructions of the installer.

You can access the packages as `mx.ODBCConnect.Client`. For more information, please see the detailed usage instructions in section 4.Using mxODBC Connect.

3.2.3 Uninstall

The Windows installer will automatically register the installed software with the standard Windows Software Setup tool.

To uninstall the server, run the Windows Software Setup tool and select the "*eGenix mxODBC Connect Client x.x.x*" entry for uninstallation. This will remove the package from your Python installation.

3.3 mxODBC Connect Client Installation on Unix

The mxODBC Connect Client is a regular Python package. It allows to connect your mxODBC compatible application to an ODBC compatible database over a network. In order to connect to a database you need to run

a properly configured mxODBC Connect Server on the machine with the target ODBC driver, usually the machine running the target database itself.

3.3.1 Prerequisites

- Python 2.5 or later installed. See *http://www.python.org* for details and download instructions.

- eGenix's mx Base extension installed. See *http://www.egenix.com/products/python/mxBase/* for details and download instructions.

- For SSL support (optional), you should either have eGenix's pyOpenSSL Distribution installed or use the Python standard lib module ssl. See *http://www.egenix.com/products/python/pyOpenSSL/* for details for details and download instructions of eGenix's pyOpenSSL distribution.

Note that using SSL encrypted communication is not allowed in all countries. Please check with your local authorities whether you are permitted to use encryption.

The client has been tested with the official Python 2.5, 2.6 and 2.7 installers. Python 2.4 and below are not supported.

3.3.2 Installation using prebuilt package archives

> **Note:**
> You may need *root* privileges to successfully complete the installation or un-installation process.

Please uninstall any existing version of mxODBC Connect Client if you have one installed (see section 3.3.5 below for details).

To reduce the number of binaries that we have to create for each release, we have adapted a new generic distribution format that works on all Python platforms: the *Prebuilt Distribution Format*.

Technically, this format is a standard Python *distutils* distribution, but with only the build/ directory and without the source tree.

After installation, you can access the packages as `mx.ODBCConnect.Client`. For more information, please see the detailed usage instructions in section 4.Using mxODBC Connect.

System-wide Installation

In order to install such a distribution, please follow these instructions:

1. Download and unzip the archive into a temporary directory

2. Change into the distribution directory

3. Run the following command using the Python interpreter with which you intend to work (this could be the default one, or an application specific one depending on your needs):

```
sudo python setup.py install
```

The distribution will then be installed into the standard directory for Python extensions of your Python installation (usually the site-packages/ subdirectory of the Python standard library directory).

To uninstall, follow the same steps as above, but use the command uninstall instead:

```
sudo python setup.py uninstall
```

User Installation

You will need to be able to `sudo` on the target machine or know the root password for the above to work. If you don't have permission to install packages as root, you can still install the distribution into a local directory, e.g. ~/lib/python2.7 by using the following installation command:

```
python setup.py install --home=/home/user/
```

This will install the distribution into the directory /home/user/lib/python/. In order to have Python see this directory and make it useable for import, you have to adjust the PYTHONPATH environment variable to include this directory, e.g.

```
export PYTHONPATH=/home/user/lib/python
```

To see all the possible installation options, run the install script using the help options:

```
python setup.py install --help
```

To uninstall, follow the same steps as above, but use the command uninstall instead:

```
sudo python setup.py uninstall --home=/home/user/
```

> **Hint:** On some Linux distributions you may get an error when using the `--home` option. In such cases, please try using the `--prefix` option instead.

3.3.3 Uninstall when using prebuilt package archives

The easiest way to uninstall the mxODBC Connect Client package is to unzip the pre-built binary package and then run:

```
sudo python setup.py uninstall
```

Depending on how you have installed the package, you have to provide additional options to the uninstall command.

If that doesn't work in your case, you can also simply remove the ODBCConnect/ subdirectory from your /…path to Python…/site-packages/mx/ directory of your Python installation (the exact location depends on your Python installation).

3.3.4 Installation using egg archives

The egg archives we provide are made available through two PyPI-style indexes which the egg tools setuptools/easy_install/pip/zc.buildout can access to automatically download and install the right egg archive.

There are two indexes, one for Python UCS2 builds:

http://downloads.egenix.com/python/index/ucs2/

and one for Python UCS4 builds:

http://downloads.egenix.com/python/index/ucs4/

Automatic Download and Installation

If you are using a Python UCS2 build, then you can install the egg archives using this command:

```
easy_install -i http://downloads.egenix.com/python/index/ucs2/ \
    egenix-mxodbc-connect-client
```

For UCS4 builds, please use this command:

```
easy_install -i http://downloads.egenix.com/python/index/ucs4/ \
    egenix-mxodbc-connect-client
```

The command line parameters for other tools such as pip are similar. Please consult their documentation for details.

Manual Download and Installation

In some cases, easy_install and other download tools cannot map the platform name to the name used in the egg archive. If you get errors during the installation, please manually download the right egg archive and then run the command directly on the downloaded egg archive:

```
easy_install \
    egenix_mxodbc_connect_client-2.0.0-py2.7.egg
```

3.3.5 Uninstall when using egg package archives

Since *setuptools* doesn't provide an uninstall command you have to manually remove the installation:

1. remove the egenix-mxodbc-connect-client.* egg directory from your Python site-packages/ directory and

2. edit the file easy-install.pth in that directory to remove the corresponding egg entry.

4.   Using mxODBC Connect

The mxODBC Connect product provides a client-server-based access to the ODBC API of ODBC managers and drivers over a network.

In order to connect to a database you need to have a properly configured mxODBC Connect Server running on the machine that provides the ODBC drivers for your database. This will typically be your database server.

4.1   Architecture of mxODBC Connect

The typical mxODBC Connect setup looks like this:

Python Application

↓

mx.ODBCConnect.Client Package

↓

(TCP/IP network with optional SSL encryption)

↓

mxODBC Connect Server service or daemon process

↓

ODBC Manager (Windows, unixODBC, iODBC)

↓

ODBC Driver

↓

Database
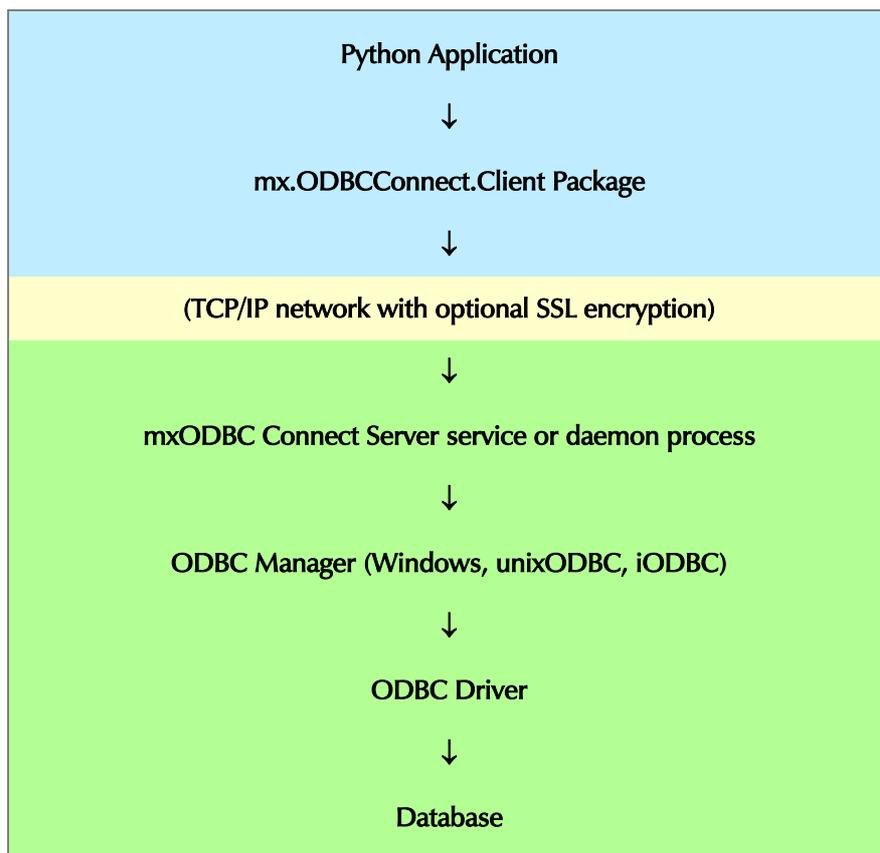
The upper blue part in the diagram executes within the process of the Python application. The green part usually runs in a separate process and usually also on a different machine.

mxODBC Connect makes your client application fairly independent of the database server. You can use the same client with 32-bit or 64-bit servers without modifications.

It is also possible to use mxODBC Connect on the same machine, e.g. if you have a 64-bit Python application that needs to use a 32-bit ODBC driver.

4.2 mxODBC Connect Client Configuration

Since the mxODBC Connect product is client-server-based, the mxODBC Connect Client will have to know where to find the corresponding mxODBC Connect Server.

The configuration data can either be stored in a client-side INI-style configuration file, or passed to the client session constructor as dictionary of dictionaries containing one dictionary per section.

4.2.1 mxODBC Connect Client Configuration File Format

If you are using a configuration file, please make sure your client application has read access to this file.

The configuration file uses an INI-file format (see section 2.4.1 on page 22 for details on the syntax) and has the following sections and options with their default values:

[Connection_Name]

Each named connection section defines a network connection used to connect to a running mxODBC Connect Server.

Multiple connections with different names can be specified to provide fail-over with multiple servers. The only requirement is that the section names contain the term "Connection" or "connection".

Examples:

[Connection_Local], [Connection_SSL], [RemoteConnection], [FailOverConnection], [CompanyVPNConnection].

Note that the order of connection sections is not preserved, so the client will try to connect to the servers in undefined order. You can define the order of connection attempts by defining the [Communication] server_connections option (see below).

host = 127.0.0.1

   IP address of the mxODBC Connect Server.

   The server must listen on this address and must be configured to accept connections from the client's IP address.

port = 6632

   Port number to connect to.

   Default port number is 6632 (IANA name *mxodbc-connect*) which is used for both plain and secure (SSL) connections and is a IANA registered port for eGenix mxODBC Connect.

   The mxODBC Connect Server must listen on this port.

   Please ensure that no firewall is blocking the communication between the client and the server.

Advanced Connection Options

You normally do not need to adjust these.

socket_timeout = None

   TCP socket timeout in seconds or None for disabling connection timeout.

   This is the length of inactivity period after the TCP connection should be dropped. You should normally not have to use this option.

Options for SSL Encrypted Connections

These options are only needed for SSL encrypted connections.

using_ssl = 0

   Setting a non-zero value enables the secure socket layer (SSL) wrapper.

You can use SSL to encrypt all communication and authenticate your clients via certificate verification (see the documentation of the server side SSL configuration "Client Certificate Access Rules" on page 27).

For SSL connections, a client certificate and private key can be provided as either file or string to enhance security and allow client authentication based on client certificates:

`client_private_key_file = client.pkey`

Name of the client's PEM-encoded private key file.

`client_certificate_file = client.cert`

Name of the client's PEM-encoded certificate file.

The server can authenticate clients by verifying their certificates. You must provide a valid and authorized certificate in order to connect to a server protected based on certificates.

`client_private_key_string = ''`

String with the client's PEM-encoded private key.

`client_certificate_string = ''`

String with the client's PEM-encoded certificate.

The server installer provides a default, self-signed certificate-key pair client.pkey and client.cert which can be used by clients.

[Communication]

Settings for network connections.

> **Note:**
> This section is only needed if you want to configure a fail-over setup for your client application.

`server_connections = defaults to the list of all defined connection sections`

This option must list connection section names as comma-separated list.

It can be used to determine the order in which the client will attempt to find a working server or to enable/disable some connection sections in the configuration file.

Default is to try all connection sections defined in the file, sorted by name.

Example:

```
server_connections = PrimaryServer, SecondaryServer
```

The client will try to connect to the mxODBC Connect Servers in the order given in this list. It will use the first successful connection. An `OperationalError` is raised if none of the configured mxODBC Connect servers allowed connections.

Listing multiple connections is useful to provide a fail-over setup. Note that the client application must catch connection errors and has to try to reconnect multiple times in order to implement a viable fail-over solution.

[Authentication]

The mxODBC Connect Server can be protected against unauthorized access using different authentication mechanisms. This section configures how authentication is handled by the server.

Note that these authentication checks are not very secure. It is generally better to use SSL connections only and implement access control via client certificate checking than relying just on authentication using a username and a password.

```
login_salt = <internal default>
```

In order to provide some extra protection when sending the login request over the network, client and server can be configured to add a salt string to the hashed login credentials.

Only set this, if you want to override the internal default or need to separate multiple mxODBC Connect installations from each other.

The salt string should not be too long and should not contain spaces. If given, the server setting for this variable must match those of the clients that want to connect to the server. The `login_salt` can be thought of as shared secret.

[Session]

This section controls the details of the communication with the server.

```
remote_module = Manager
```

Name of the mx.ODBC subpackage to be used to connect the database on the **server side**, regardless of the OS the client runs on.

Possible values: `Windows`, `iODBC`, `unixODBC` and `Manager`

The default value `Manager` will have the server will use the default ODBC manager on the server side. This allows the client to be mostly independent of the server's configuration.

`enable_compression = 1`

Network communication compression.

Setting this variable to 1 will enable compression of TCP packets sent by the client, setting the variable to 0 causes compression to be disabled.

Compression is enabled per default, in order to reduce network traffic and enhance roundtrip times.

On very fast networks or local connections you may want to disable compression for enhanced performance. We have found that even on Gigabit Ethernet networks, enabling compression does provide a performance increase.

`compression_ratio = 2`

Compression ratio to use for network communication compression.

Valid values are 1 (least compression, fast) - 9 (best compression, slow).

The default value of 2 is a good compromise for fast networks.

You may want to experiment with the setting to tune it for best performance on your network.

In some setups, e.g. fast server and slow clients, it may be wise to use different compression ratios for clients and servers. The server setting affects packets sent from the server to the client, whereas the client setting affects packets sent from the client to the server.

`max_chunk_length = 64000`

Maximum chunk length for TCP read/write operations.

You normally don't have to change this value.

`receive_timeout = 10`

Timeout for one TCP receive operation in seconds.

You will normally not need to change this value.

`send_timeout = 10`

Timeout for one TCP send operation in seconds.

You will normally not need to change this value.

[Logging]

This section defines the details of logging output.

```
log_level = mx.Log.SYSTEM_IMPORTANT
```

> Log level. See mxLog for details.

```
log_file = client.log
```

> This is the name of the log file to use.
>
> Please make sure your client application has write permission to this file (and possibly the directory).

[Integration]

This section defines configuration details needed for integrating mxODBC Connect Client with third party software.

```
ssl_module = no default value
```

> Defines which SSL module mxODBC Connect Client should try to import and use on the client side. Possible values are `ssl` or `pyOpenSSL`.
>
> When not set, mxODBC Connect Client will first try to import the pyOpenSSL module and fallback to the Python standard library ssl module (available in Python 2.6 and later), if this doesn't work.

```
gevent = 0
```

> mxODBC Connect Client comes with gevent support. If you are using the gevent library, you can set this setting to 1 in order to enable mxODBC Connect Client's gevent support. It will then integrate with the gevent library and use the asynchronous versions of the socket and ssl modules instead of the regular ones.

The [Integration] section was added in version 2.0.

4.2.2 Configuration Dictionary Format

The mxODBC Connect Client session constructor `ServerSession` takes a parameter `config_data` which can be used to configure the session without requiring installation of a client-side configuration INI file or to override certain settings from the configuration file with new values.

The `config_data` dictionary must provide the same data as an INI file, but prepared as dictionary of dictionaries, with one dictionary per INI-section, e.g.

```
config_data = {
    'Logging': {
        'log_file': 'client.log',
```

```
    },
    'Communication': {
        'server_connections': 'Connection_SSL, Connection',
    },
    'Connection': {
        'host': '192.168.1.100',
        'port': 6632,
    },
    'Connection_SSL': {
        'host': '192.168.1.100',
        'port': 6632,
        'using_ssl': 1,
    }
}
```

For details on section names and options, please see section 4.2.1 mxODBC Connect Client Configuration File Format. For details on the `ServerSession` constructor API, please see section 5.2 Multi-Threaded Applications.

If both `config_file` and the `config_data` dictionary are given on the `ServerSession` constructor, the values from `config_data` are merged into the values read from the configuration file or override them.

4.2.3 mxODBC Connect Client Configuration Hints

Since the mxODBC Connect Server runs on the server machine, the client applications cannot or should not always know which ODBC manager the server machine uses as default.

For this reason, mxODBC Connect Server provides a generic interface to the server's default ODBC manager. The corresponding mxODBC sub-package is called `mx.ODBC.Manager`.

By configuring all mxODBC Connect Clients to use this package as server side package, you make sure that the clients will always use the default ODBC manager on the server side.

Because this is a useful setting, we have made it the default in the client configuration. If you want to make it explicit, simply configure the clients to use the Manager module:

```
[Session]
remote_module = Manager
```

4.3 mxODBC Connect Client Example

First, you have to setup a working mxODBC Connect Server on the machine that has the ODBC drivers installed (see above).

4.3.1 Client Configuration

After you have a working server, you'll have to create a client side configuration file.

Contents of the example connect-config.ini:

```
[Logging]
    log_file = client.log

[Server_Connection]
    host = 192.168.1.100
    port = 6632
    using_ssl = 0
```

The client in the above example will connect to a Windows based mxODBC Connect Server which listens on 192.168.1.100:6632 for plain text (unencrypted) connections..

4.3.2 Connecting to the mxODBC Connect Server

Your stand-alone mxODBC based application usually connects to the database like this:

```
import mx.ODBC.Windows as ODBC
```

and then uses the ODBC object to reference the mxODBC API.

In order to use the mxODBC Connect Client you have to successfully connect to an mxODBC Connect Server first to get a reference to an object implementing the mxODBC API:

```
from mx.ODBCConnect.Client import ServerSession

session = ServerSession('connect-config.ini')
ODBC = session.open()
```

Creating a `ServerSession` instance connects to the mxODBC Connect Server. The `ServerSession` instance represents your connection to the mxODBC Connect Server, so you have to keep a reference to the object or your connection will be lost.

The `.open()` method returns an efficient proxy object which implements the same API as mxODBC's subpackages have. The subpackage you are proxying requests to depends on the client configuration setting `[Session]` `remote_module`. It defaults to `mx.ODBC.Manager` which is an alias to the server platform's default ODBC manager and should be a reasonable choice in most cases. Please see the *mxODBC documentation* for details on how the `mx.ODBC.Manager` alias is chosen.

After this initial setup has been done, you can use the ODBC object as if you were running the application on the mxODBC Connect Server machine, e.g.

```
connection = ODBC.DriverConnect('DSN=myDSN;UID=user;PWD=pwd')
cursor = connect.cursor()
cursor.execute('select * from mytable')
results = cursor.fetchall()
cursor.close()
connection.close()
```

Once you are done using the ODBC session object, you should call the `.close()` method on the session object in order to free the resources on the server side and close any still open database connections associated with the client:

```
ODBC.close()
```

The `ServerSession` object will also close itself at garbage collection time (ie. when all references to it have been removed from Python namespaces), however, it is not always clear when this happens due to the way Python's garbage collection works, so closing the session explicitly is the preferred way to close the session.

Storing ServerSessions as module globals

Please note that if you store the `ServerSession` object as module global, the object will likely only be garbage collected at Python interpreter shutdown time, i.e. when exiting the application.

Since Python cleans up the various module namespaces in more or less random order, the implicit closing of the session may not succeed: Python may have already removed part of the mxODBC Connect Client libraries needed to communicate with the server.

If you use the `ServerSession` object as module, please register its `.close()` method as Python `atexit` function. Python will then call the `.close()` method just before starting to shutdown all modules when exiting the application:

```
# Register atexit function to make sure that the session object
# gets closed before the module gets destroyed.
import atexit
atexit.register(ODBC.close)
```

Note that mxODBC Connect Server will automatically free up resources on the server side if it detects a broken connection to the client. Even without successfully calling the `.close()` method, the database connections will get closed on the server side.

However, there is a slight delay compared to the explicit approach, since the server only checks connections in regular intervals (usually every 2 seconds, see `[Activity] check_interval` in the server configuration file documentation on page 31).

4.3.3 Exception Handling

Exception classes originally imported from `mx.ODBC.Error` will have to be imported from `mx.ODBCConnect.Error` when using the mxODBC Connect Client.

This may require slight modifications to your client application.

All exception classes imported from `mx.ODBCConnect.Error` are subclassed from the same built-in exception classes as their original counterparts from `mx.ODBC.Error`, so generic except clauses should work as expected.

See the sections below for additional features, differences and limitations.

4.4 Testing

To thoroughly test your mxODBC Connect setup, you can use the new mxODBC test script **test.pyc** on the client side, which has the ability to run tests against the mxODBC API through an mxODBC Connect Server.

You have to specify *ODBCConnect* as package and pass the name of your test configuration file to test.pyc:

```
python mx/ODBCConnect/Misc/test.pyc \
   --package=ODBCConnect \
   --dsn="DSN=…;UID=…;PWD=…" \
   --client-config=client-config.ini \
   --client-log=client.log
```

(all on one line and without the backslashes ("\"))

This will test a lot of mxODBC and database features, many of which are only supported by a few databases, so expect quite a few "not supported" messages.

> **Note:**
> test.pyc currently only supports anonymous server logins. This may change in future versions of the mxODBC Connect Client.

test.pyc Options

`--package`

Name of the mxODBC package to test.

`--dsn`

ODBC datasource connection string.

`--client-config`

Location of the mxODBC Connect Client configuration file.

`--client-log`

Location of the mxODBC Connect Client log file. This is optional. The script defaults to logging everything to `stderr`.

For more information on the parameters, run test.pyc with option '--help'. It will then print a help screen with available options.

> **Note:**
> The mxODBC subpackage to be used will be determined by the [Session] `remote_module` setting in you test configuration file. This is `'Manager'` by default, which means the default ODBC manager installed on the server side.

5. mxODBC Connect Client Python API

The mxODBC Connect Client provides a way to easily access the mxODBC package API of the mxODBC Python extension on the mxODBC Connect Server.

All the network communication, proxying and error handling is done transparently by the mxODBC Connect client-server logic, to make using the client API as easy as using the stand-alone version of mxODBC.

Applications that were written for mxODBC should not require significant changes when porting them to mxODBC Connect.

5.1 API Design

Since mxODBC Connect works in a client-server setup, the client will first have to initialize a server session. This is done by creating a *ServerSession* object.

The `ServerSession` object maintains the configuration information and deals with the network communication between the client and the mxODBC Connect Server.

Since mxODBC supports multiple subpackages to implement access to different ODBC manager and drivers, the `ServerSession` object allows connecting the session to one of these packages.

The default package is defined by the `remote_module` setting in the [Session] section of the configuration file or settings. If not given, mxODBC Connect will use the `mx.ODBC.Manager` package which always defaults to the standard ODBC manager on the platform where mxODBC Connect Server is installed, e.g. `mx.ODBC.Windows` on Windows, `mx.ODBC.iODBC` on Mac OS X and one of `mx.ODBC.unixODBC` or `mx.ODBC.iODBC` on other Unix platforms.

In order to connect a `ServerSession` object to the remote mxODBC package on the server side, an application must call the `.open()` method on the object to open and initialize the connection to the server.

mxODBC Connect Client will then return a module proxy object that makes the package's API available on the client side. This works very much like an import in Python, e.g. instead of writing:

```
from mx.ODBC import Manager as ODBC
```

in the stand-alone version of mxODBC, you'd write:

```
from mx.ODBCConnect.Client import ServerSession

session = ServerSession(config_file='conf.ini')

ODBC = session.open()
```

The ODBC module proxy object will then work in a nearly identical way as the ODBC module from the stand-alone version of mxODBC.

If you have multiple connections defined in your mxODBC Connect Client configuration, then `.open()` will try the connections in the order specified by the `[Communication] server_connections` parameter.

In the following sections, we describe APIs that are special to the mxODBC Connect version of mxODBC.

Once you have a module proxy object, you can use the standard mxODBC connection and cursor APIs described in the *mxODBC User Manual and Reference Guide*.

5.2 Multi-Threaded Applications

mxODBC Connect works well in multi-threaded applications and is written in a thread-safe way.

In order to make the best use of the available technology, you should consider the options that you have to manage connections between threads as outlined in the following sections.

Note that mxODBC Connect itself will not start new threads, so **you can safely use it in non-threaded applications as well.**

5.2.1 Recommended Setups

If you intend to use mxODBC Connect with a multi-threaded application, you have two possibilities:

1. use a single `ServerSession` for the application, connected to the mxODBC Connect Server, and have one or more database connections open per thread of your application (this does not work for SSL connections), or

2. have one `ServerSession` object per thread that needs to connect to the database (this is the **preferred method**).

Please do not try to **share database connections between threads**. This is not supported by the mxODBC Connect Client and not needed due to the way mxODBC Connect works.

Also note that **for SSL connections, you can only use option 2**, since the OpenSSL library used by mxODBC Connect does not support sharing SSL connections between threads.

Both options allow using the mxODBC Connect Server from the multiple threads that open database connections.

However, there is one important difference: all connections opened by a single `ServerSession` are mapped to a single thread on the server. As a result, the operations on the session are serialized on the server and thus do not run in parallel.

For applications that run many quick queries, the difference will likely not be noticeable, but if you have long running queries, you should definitely choose the second method in order to have full flexibility of running the queries in parallel.

The difference in amount of resources used by the two methods is negligible. You should only consider using the first method or a combination of the two in case you are planning to have many threads running in parallel, each connected to a database.

5.2.2 Logging

If you intend to use custom `mx.Log` logging objects (via the logging parameter in `ServerSession` objects), please make sure that you share the logging object if you want to log to the same log file - not the log file itself. Otherwise, you could end up seeing mangled output in the log file.

5.3 gevent Support

Tested with gevent 1.0.1 and greenlet 0.4.2.

mxODBC Connect Client can optionally integrate with the *greenlets* via the Python package *gevent* and using the *libevent* polling library.

To enable gevent support for the mxODBC Connect Client, please enable the integration setting in the client side configuration:

```
[Integration]
gevent = 1
```

mxODBC Connect Client will then use gevent APIs for communicating with the server side, allowing other greenlets to run asynchronously while the client waits for the server response.

5.3.1 Import Order

For best compatibility, please import the gevent package before importing the mxODBC Connect Client into your Python application.

5.3.2 gevent Monkey-Patching

The client does not require enabling the gevent monkey patching features, nor does it enable these itself. We have tested mxODBC Connect Client in a gevent monkey-patched environment, but recommend using gevent APIs directly rather than through the monkey patched setup.

5.4 mxODBC Connect Client ServerSession Object

A *ServerSession* object manages the connection of a client application to the mxODBC Connect Server.

It provides all the necessary networking logic to proxy requests to mxODBC package module APIs to the server side in an efficient and reliable way.

Module:

`mx.ODBCConnect.Client.ServerSession.ServerSession`

The class is also available directly via the `mx.ODBCConnect.Client` module.

<u>Usage Example:</u>

```
from mx.ODBCConnect.Client import ServerSession

# Setup a server session
session = ServerSession(config_file='client-config.ini')

# Connect to the mxODBC Connect Server
server = session.open()

# Connect to a database
connection = server.DriverConnect('DSN=...;UID=...;PWD=...')
```

Object Constructor:

```
ServerSession(config_file=None, config_data=None,
    logging=None)
```

Initialize and configure an mxODBC Connect server session.

`config_file` can be given to specify a configuration file. If no configuration file is specified, defaults are used instead.

`config_data` can be given as dictionary to override settings from the `config_file` or the defaults. The dictionary has to include one dictionary per INI section of the configuration that should be overridden. It is possible to provide all configuration parameters via `config_data`.

`logging` may be given to have the session use a different `mx.Log` object. The default log object is setup using the configuration details from the `config_file` and/or `config_data`.

Base class(es): object

Object Attributes:

`.closed = True`

State of the session.

True or False (1 or 0), depending on whether the session is connected or not.

`.server_version = ''`

Version string of the mxODBC Connect Server.

This is only available after opening the session.

`.session_id = None`

Session ID string.

Object Methods:

`.close()`

Closes the session.

This method is automatically called when the session object is garbage collected.

You can call this method to explicitly close the connection before deleting the session object.

`.open(username='', password='', module_name=None, session_id=None)`

Connect to the first available and working server connection listed in the mxODBC Connect Client configuration. The method raises an `mx.ODBCConnect.Error.OperationalError` if no connection can be established.

Returns a module proxy object that exposes the mxODBC package API of the `module_name` package of mxODBC on the server side.

`username` and `password` must be given if the server uses user authentication. They must be set to the user authentication credentials defined on the server side.

`module_name` defaults to the mxODBC package name defined in the client configuration parameter `[Session] remote_module`. This is usually set to `'Manager'`, so that the server's platform default ODBC manager is used.

`session_id` may be given to reestablish the connection to a server session, e.g. in case the network was down or unavailable for only a short period of time.

5.5   mxODBC Connect Client Errors

All mxODBC Connect errors raised on the client side are available through the `mx.ODBCConnect.Error` module.

The errors are grouped into errors which originate on the server side and get reraised on the client side (*Server Side Errors*) and ones which are raised

by the session logic (*Session Errors*). The latter are mostly related to network problems. A client application should try to catch these errors and issue a reconnect.

The Server Side Errors are also available via the mxODBC connection object as attributes (just like they are in the stand-alone mxODBC product).

5.5.1 Server Side Errors

`Error`

Baseclass for all other exceptions related to database or interface errors.

You can use this class to catch all errors related to database or interface failures. `error` is just an alias to `Error` needed for DB-API 1.0 compatibility.

Error is a subclass of `exceptions.StandardError`.

`Warning`

Exception raised for important warnings like data truncations while inserting, etc.

Warning is a subclass of `exceptions.StandardError`. This may change in a future release to some other baseclass indicating warnings.

`InterfaceError`

Exception raised for errors that are related to the interface rather than the database itself.

`DatabaseError`

Exception raised for errors that are related to the database.

`DataError`

Exception raised for errors that are due to problems with the processed data like division by zero, numeric out of range, etc.

`OperationalError`

Exception raised for errors that are related to the database's operation and not necessarily under the control of the programmer, e.g. an unexpected disconnect occurs, the data source name is not found, a transaction could not be processed, a memory allocation error occurred during processing, etc.

IntegrityError

Exception raised when the relational integrity of the database is affected, e.g. a foreign key check fails.

InternalError

Exception raised when the database encounters an internal error, e.g. the cursor is not valid anymore, the transaction is out of sync, etc.

ProgrammingError

Exception raised for programming errors, e.g. table not found or already exists, syntax error in the SQL statement, wrong number of parameters specified, performing operations on closed connections etc.

NotSupportedError

Exception raised in case a method or database API was used which is not supported by the database, e.g. requesting a .rollback() on a connection that does not support transaction or has transactions turned off.

This is the exception inheritance layout:

```
StandardError
|__Warning
|__Error
   |__InterfaceError
   |__DatabaseError
      |__DataError
      |__OperationalError
      |__IntegrityError
      |__InternalError
      |__ProgrammingError
      |__NotSupportedError
```

5.5.2 mxODBC Connect Error Module

Note that unlike mxODBC, the exception classes are not available at the package module top-level, ie. `mx.ODBCConnect.ProgrammingError` does not work.

Instead you have to refer to the exception classes via the `mx.ODBCConnect.Error` module, e.g.

```
from mx.ODBCConnect.Error import ProgrammingError
```

or

```
import mx.ODBCConnect.Error
try: … except mx.ODBCConnect.Error.ProgrammingError: …
```

5.5.3 Session Errors

`ConfigurationError`

Raised for errors found in the client or server configuration files or data.

`ConnectionFailureError`

Raised when connection closed prematurely and in other cases.

`ODBCConnectError`

This is the base class for all mxODBC Connect related errors, e.g. ones raised due to protocol or policy errors.

It is a subclass of the server error `InterfaceError` to allow catching the error in DB-API compatible applications which were not specifically written for mxODBC Connect.

`PolicyViolationError`

Error caused by configuration limits on the server side.

`ProtocolError`

Error in the mxODBC Connect protocol, e.g. due to a version mismatch between client and server.

`TimeoutError`

Error due to a connection or server timeout.

5.6 mxODBC API

All other aspects of creating connections and cursors can be taken straight from the stand-alone version of mxODBC.

Please see the *mxODBC User Manual and Reference Guide* for details on the connection and cursor APIs and specific suggestions for many different common database backends.

The differences between the mxODBC Connect Client and the stand-alone version of mxODBC are described in the next section.

6. Differences between mxODBC and mxODBC Connect

The most important difference between the stand-alone product mxODBC and the client-server product mxODBC Connect is the ability to separate the requirements regarding the ODBC driver setup and configuration from the requirements of client application using mxODBC.

With mxODBC Connect, database access becomes mostly independent of the differences between the server running the database and the client machine running your application. They may have different number of CPUs, bit architectures, byte ordering (LSB/MSB) and operating systems (Windows/Linux).

eGenix has tried hard to make porting of mxODBC applications to mxODBC Connect as easy as possible. Most features available in the stand-alone mxODBC are also available in mxODBC Connect. However, there are a few minor differences between direct and networked access to mxODBC:

6.1 Additional Features in mxODBC Connect

6.1.1 Improved portability

The mxODBC Connect Client can be installed on any platform that is supported by the *eGenix mx Base Distribution* package - which is pretty much any platform that Python itself runs on.

There is no need to find a suitable ODBC driver for the platform on which you intend to install the client. This removes one of the major obstacles in getting mxODBC to run on more exotic platforms.

If you want to use encryption, you will also need the Python standard library module `ssl`, which is available in Python 2.6 and later, or our *eGenix pyOpenSSL Distribution* package. However, this is not essential for working with mxODBC Connect.

6.1.2 Improved data type support

Since the mxODBC Connect Server always runs on Python 2.7, the decimal and datetime modules are always available on the server side. This allows clients still running on Python 2.5 or 2.6 to communicate with the database using types from these packages.

The Python **datetime module** and **decimal module** can be fully utilized with Python 2.5 and 2.6 clients.

6.1.3 Improved Scalability

You can separate your client application and database server for improved performance and scalability, e.g. to work around problems with the Python Global Interpreter Lock (GIL)[4].

They can reside on different physical or virtual machines or just run on different CPU cores.

6.1.4 Asynchronous Execution Support using gevent

mxODBC Connect Client 2.0 and later support the *gevent* module for running tasks asynchronously without using threads or by combining asynchronous execution with threads. This allows for better scaling of Python client side applications, especially on multi-core machines.

Please see section 5.3 gevent Support for details.

6.1.5 Automatic Fail-over

You can list multiple servers in your client configuration. Your `ServerSession` will connect to the first working server in the configured list.

You can also provide exception handlers for automatic reconnection on connection lost errors and you have full control over the order in which the connections attempts are done.

---

[4] The Python Global Interpreter Lock (GIL) serializes access to the Python interpreter: only one thread can execute Python code at a time.

6.1.6 Data compression

mxODBC Connect uses data compression for communication between the client and server. This reduces the network traffic load and results in faster roundtrips.

As a result, using mxODBC Connect is often faster then using mxODBC together with a client side ODBC driver.

6.2 Differences and Limitations

6.2.1 Parameter Data Types

Since mxODBC Connect runs all database operations on the server side, it has to transfer the Python objects passed as parameters to the `cursor.execute*()` methods (and other methods accepting arbitrary objects) over the network in serialized form.

This operation will only succeed for basic pickleable Python types (Unicode, string, numbers, etc.) as well as eGenix mxDateTime instances, since the server only provides support for these types.

Other objects types, such as user-defined subclasses, cannot be unserialized on the server side and thus may result in mxODBC Connect exceptions to be raised.

In order to work around this limitation, please make sure that the parameter values you pass to the `cursor.execute*()` methods only use supported data types.

No support for Python 2.7 memoryviews

Unfortunately and unlike many other basic Python types, Python 2.7 *memoryviews* cannot be pickled. This is a limitation of Python, not mxODBC Connect.

As a result, they cannot be used as parameters to `cursor.execute*()` methods and are not available for passing data to the server database.

6.2.2 Garbage collection and closing of connections / cursors

mxODBC Connect Client manages a cache of objects in order to increase performance and provide more reliability.

Due to this cache, garbage collection of e.g. database connection or cursor objects may not directly result in the objects to get implicitly closed.

This may result in a situation where e.g. connections are kept open on the server side longer than necessary and even result in the application hitting a database connection license limit on the server more often than necessary.

You can easily prevent this, by explicitly closing cursor and connection objects after use.

Example:

```
connection = session.DriverConnect(…)
cursor = connection.cursor()
# do some work with cursor
cursor.close()
cursor = None
connection.close()
connection = None
```

6.2.3 Exceptions

Exception classes must be imported from `mx.ODBCConnect.Error` instead of `mx.ODBC.Error`, which may require slight modification to existing application code. Unlike in mxODBC, the exception classes are not available via the top-level `mx.ODBCConnect` module, ie. `mx.ODBCConnect.ProgrammingError` does not resolve to the `ProgrammingError` exception class.

Note that all exceptions are subclassed from the same built-in exception classes as their mxODBC equivalents, so generic error handlers will work without modifications.

The client should be modified to catch the new exceptions of the mxODBC Connect Client API, such as loss of network connection. However, this is only required, if you need advanced connection handling and automatic fail-over.

6.2.4 Converter Functions

Converter functions are not supported. They may be supported by a later version of mxODBC Connect.

6.2.5 Error Handlers

Error handlers are not fully supported. They may be supported by a later version of mxODBC Connect.

It is possible to register an error handler with mxODBC Connect Client, but the exceptions will still always be raised. This is mainly due to the fact that error handlers run on the client side.

Database Warnings

For the most common case of using error handlers, ignoring database warnings, you can use the `.warningformat` connection/cursor attribute which allows choosing from different mechanisms to e.g. ignore warnings on the server side.

Example:

```
from mx.ODBCConnect.Client import ServerSession
# Setup a server session
session = ServerSession(config_file='client-config-windows.ini')
# Connect to the mxODBC Connect Server
server = session.open()
# Connect to a database
connection = server.DriverConnect('DSN=sqlserver2008;UID=sa')
# Ignore warnings issued by the database (e.g. for context
# switches)
connection.warningformat = server.IGNORE_WARNINGFORMAT
```

Please see the *Database Warning* section in the *mxODBC documentation* for more details.

6.2.6 Server-side Exceptions

When printing exceptions raised on the server-side, the client will only display a partial traceback, containing the client side traceback information. All other exception information is preserved.

Note that server side exceptions are logged by mxODBC Connect Server - including their full traceback.

This limitation can also be considered a feature, since it prevents accidental leakage of confidential information from the server to the client side.

6.2.7 RowFactory Helper Module

As part of the upgrade to the mxODBC 3.3 API, mxODBC Connect 2.1 also supports the new `cursor.row` and `cursor.rowfactory` attributes. Just like mxODBC 3.3, mxODBC Connect also comes with a new helper module to simplify use of these attributes.

In mxODBC, the module is available as `mx.ODBC.Misc.RowFactory`. mxODBC Connect comes with the same module, but as `mx.ODBCConnect.Misc.RowFactory`.

Unlike the mxODBC subpackage modules, which are accessed via a proxy to the server side, the `RowFactory` module exists on the client side as importable module, even without server connection. To stay compatible to the mxODBC API, the module is additionally available via the subpackage module object as `.RowFactory` attribute.

All row class processing happens on the client side, so the added overhead is for setting up the row classes is minimal.

6.2.8 Using the cursor.row attribute

With the `RowFactory` module available, it is easy to create custom row classes for a given result set.

The only requirement is having a cursor with prepared or executed query (so that the `cursor.description` information is available).

Example:

```
from mx.ODBCConnect.Client import ServerSession

# Setup a server session
session = ServerSession(config_file='client-config-unix.ini')

# Connect to the mxODBC Connect Server
server = session.open()

# Connect to a database
db = server.DriverConnect('DSN=…;UID=sa;PWD=…')
print 'Connected to %s %s' % (db.dbms_name, db.dbms_version)

# Create cursor
cursor = db.cursor()
```

```
# Create a cursor with information about the result set
cursor.execute('select * from mytable')

# Create the row class
MyTableRow = server.RowFactory.TupleRowFactory(cursor)

# Fetch the data using MyTableRow objects
cursor.row = MyTableRow
rows = cursor.fetchall()
```

In the above example, mxODBC Connect will fetch all rows as `MyTableRow` instances, which are Python tuple subclasses with added named based access to the columns, so that you can write `row[0]`, just as well as `row['id']` or `row.id` (assuming that the first column of the result is named '*id*'.

Pickling Dynamic Row Classes

Note that *MyTableRow* instances are not pickleable. The reason is that pickle cannot associate a module and attribute with them to place into the pickle information.

If you want to pickle such row objects, you have to add the needed information to the generated class:

```
# Adjust the MyTableRow class so that pickle can find the
# right module and class
MyTableRow.__name__   = 'MyTableRow'
MyTableRow.__module__ = __name__
```

In this example, __name__ refers to the module which holds the attribute to the row class.

6.2.9   Using the cursor.rowfactory attribute

If you don't know the result set layout in advance, you can use the row class factory functions of the RowFactory module to have them build the row classes for you when the first row is fetched from the database:

Example:

```
from mx.ODBCConnect.Client import ServerSession

# Setup a server session
session = ServerSession(config_file='client-config-unix.ini')

# Connect to the mxODBC Connect Server
server = session.open()

# Connect to a database
db = server.DriverConnect('DSN=…;UID=sa;PWD=…')
print 'Connected to %s %s' % (db.dbms_name, db.dbms_version)
```

```
# Create cursor
cursor = db.cursor()

# Fetch the data using MyTableRow objects
cursor.rowfactory = server.RowFactory.ListRowFactory

# Create result set
cursor.execute('select * from mytable')
rows = cursor.fetchall()
```

In the above example, the result set layout information is not available at the time the `cursor.rowfactory` is set.

mxODBC Connect will call the `cursor.rowfactory` function with the cursor as first argument just before fetching the first row of the result set.

Using `cursor.rowfactory` results in much better performance compared to other solutions, which build e.g. `namedtuples` for each and every row, since the class building only has to be done once per result set.

6.2.10 Using iterators/generators with cursor.executemany()

mxODBC 3.3 introduced support for iterators/generators as argument for `cursor.executemany()`.

Since iterators and generators are polled for more information and cannot be pickled, mxODBC Connect processes the data in chunks by reading up to 2048 data rows from the iterator/generator and sending the request to the server side.

To make this customizable, mxODBC Connect provides a special extra parameter for `cursor.executemany()`, which is not available in mxODBC:

```
cursor.executemany(sqlcmd, batch, direct=0,
    parametertypes=None, chunksize=2048)
```

> …
>
> For iterators and generators, a keyword parameter `chunksize` may be given to specify the number of rows to send to the server in a single chunk. It defaults to 2048 rows.

When seeing an iterator or generator as batch parameter, mxODBC Connect will read data from it in chunks of `chunksize` and then process the `.executemany()` call on this chunk in one request to the server side.

7. Troubleshooting

Please always consult the FAQs before contacting eGenix Support (*support@egenix.com*).

7.1 Frequently Asked Questions (FAQ)

This section lists frequently asked questions regarding mxODBC Connect.

7.1.1 Where can I find the server.log file on Windows ?

If you have installed the mxODBC Connect Server tray icon helper, you can open this file using the tray icon's menu entry *Show Log File*.

The server.log file is located in the C:\<documents and settings>\<all users>\<application data>\eGenix.com\mxODBC Connect Server\ directory (the exact names of the path components depend on your Windows installation).

7.1.2 Where can I find the server.log file on Unix ?

It is located in the home directory of the *mxodbc* user, usually /opt/eGenix/mxODBC-Connect-Server/.

7.1.3 The Windows installer stops with a message that a file cannot be installed

This sometimes happens when you reinstall or update the mxODBC Connect Server. Please try the following:

- Make sure that you have shutdown a possibly running mxODBC Connect Server using the tray applet.

- Close the mxODBC Connect Server tray applet.

- Make sure that you have no running processes that start with "*mxODBC-Connect*".

- Click on "Retry" in the installer message dialog.

If the problem persists, you will have to cancel the installation and restart the system, before retrying the installation.

Since the mxODBC Connect Server runs as Windows service, it is possible that a system process still references it or one of its DLLs.

7.1.4 mxODBC Connect Server for Windows doesn't start

If you have correctly installed the server licenses, but the server fails to start, please have a look at the server.log file. See FAQ entry 7.1.1 for details on how to open this log file.

The log should provide an explanation of what caused the startup failure.

Please make sure that:

- the configuration file doesn't have any errors, e.g. duplicate section names, illegal values, mistyped option names, etc.

- the license files, configuration file and certificates are readable by the service user

7.1.5 mxODBC Connect Server for Unix doesn't start

If you have correctly installed the server licenses, but the server fails to start, please have a look at the server.log file. See FAQ entry 7.1.2 for details on where this file is stored.

If the log file mentions a missing libodbc.so.1 or libiodbc.so.2 file , then the server cannot find your ODBC manager installation.

Please check the following:

- You have one of iODBC or unixODBC installed.

- You have installed the correct version of the eGenix mxODBC Connect Server for your platform, ie. the x86 version for a 32-bit Linux and the x64 version for a 64-bit Linux.

- The dynamic linker (usually *ld.so*) is setup to find the shared libraries of the installed ODBC manager; `ldconfig -p` should list the libodbc.so or libiodbc.so files.

- The *mxodbc* user account has permission to access the shared library files.

7.1.6 Importing exceptions from mx.ODBC.Error fails (no such module)

You have to import the mxODBC related exception classes from `mx.ODBCConnect.Error` instead of `mx.ODBC.Error` when using mxODBC Connect Client.

Simply search&replace your imports and insert `'Connect'` in the appropriate places.

7.1.7 Exceptions are not caught as expected at client side

You have to import the mxODBC related exception classes from `mx.ODBCConnect.Error` instead of `mx.ODBC.Error` when using mxODBC Connect Client.

All exceptions imported from `mx.ODBCConnect.Error` are subclassed from the same built-in exceptions as the original mxODBC ones.

7.1.8 Client cannot connect to the mxODBC Connect Server.

- Ensure, that your mxODBC Connect Server is configured correctly and the service or daemon runs without a fatal error.

- Check the server logs for connection attempts.

- Ensure that no firewall block the connection on either side.

- Check your client certificate if the server has client certificate verification turned on.

7.1.9 Converter function has been set, but not called.

Converter functions are not supported in the current release of mxODBC Connect. They might be supported by a later release.

Please report your problem to eGenix.com to let us know about your requirements.

7.1.10 Error handlers don't seem to work.

Exception will always be raised, even if the error handlers don't reraised them.

This is due to the fact that error handler must run on the client side and therefore cannot influence how the mxODBC Connect Server handles in-process error situations.

7.1.11 Printing exception tracebacks does not include the server side.

Server-side exceptions with full tracebacks can be read in the server logs if needed, e.g. to track down problems related to database ODBC drivers.

7.1.12 InterfaceError: Connection limit exceeded. Your license allows 20 physical database connections.

This error is the result of having too many physical database connections open on the server side.

Database connections on the server side are opened and closed following the connect and close calls on the client side. However, in some cases, e.g. due to errors in the client side application, these may not get called and result in the connections to stay open on the server side.

Please always explicitly close your client side database connections using the connection object's `.close()` method.

Another situation where this may happen can occur when not explicitly closing the `ServerSession` object on the client side or having this close process fail due to network problems.

The server will only check client connections every few seconds, so the connections may be kept open on the server side even though the client application has already terminated.

If you frequently start client applications which don't close their `ServerSession` object, this may result in the number of concurrently open connections to reach the license limit, giving the above error message.

Please always explicitly close the `ServerSession` object using its `.close()` method.

7.1.13 Error "Maximum number of sessions reached." with unlimited connections license

If you are receiving errors from the server mentioning a maximum number of sessions being reached, even though you have a license with unlimited number of connections installed, you will have hit a configurable limit in the server configuration that is intended to prevent denial of server attacks.

To resolve the issue, please change the server configuration to use a higher limit for the parameter *max_sessions* in the *[Activity]* section of the configuration. The default value is set to 400 sessions.

8. Hints & Links to other Resources

8.1 More Sources of Information

There are several resources available online that should help you getting started with ODBC. Here is a small list of links useful for further reading:

Microsoft MDAC Site

Microsoft is constantly developing new forms of database access. For a close up on what they have come up recently take a look at their ODBC site. Note that they now call their ODBC SDK "Microsoft Data Access Components SDK" (MDAC). It does not only focus on ODBC but also on OLE DB and ADO.

Note: If you are not happy about the size of the SDK download (over 31MB), you can also grab the older 3.0 SDK which might still be available from a FTP server. Look for "odbc3sdk.exe" using e.g. FTP Search.

Microsoft also supports a whole range of (desktop) ODBC drivers for various databases and file formats. These are available under the name "ODBC Desktop Database Drivers" (search the MS web-site for the exact URL) [wx1350.exe] and also included in the more up-to-date "Microsoft Data Access Components" (MDAC) archive [mdac_typ.exe].

Microsoft ODBC Portal

This portal page has a few interesting links into the Microsoft ODBC site. If you're looking for the latest SQL Server or Oracle ODBC drivers this is the place to look first.

ODBC Documentation

The ODBC documentation is included in the free MS MDAC SDK which you can download from their *ODBC site*.

SQLSummit List of ODBC drivers

A collection of available ODBC driver packages. This should be the first place to look in case you are searching for OBDC connectivity to your database.

9.   Support

eGenix.com is providing commercial support for this package, including adapting it to special needs for use in customer projects.

If you are interested in receiving information about this service please see the *eGenix.com Support Conditions*.

10. History & Changes

Please visit the *product page* on the eGenix.com website for the list of changes.

11. Copyright & License

© 1997-2000, Copyright by IKDS Marc-André Lemburg; All Rights Reserved. mailto: *mal@lemburg.com*

© 2000-2014, Copyright by eGenix.com Software GmbH, Langenfeld, Germany; All Rights Reserved. mailto: *info@egenix.com*

This software is covered by the **eGenix.com Commercial License Agreement**, which is included in the following section 11.1. The text of the license is also included as file "LICENSE" in the package's main directory.

The software also includes third party software which is covered by other licenses. The text of those licenses is included in the following section xxx.

> Please note that using this software is **not free of charge**. You may use the software during an evaluation period as specified in the license, but subsequent use requires the ownership of a "Proof of Authorization" which you can buy online from *eGenix.com*.

Please see the *eGenix.comWebsite* for details about the license ordering process.

By downloading, copying, installing or otherwise using the software, you agree to be bound by the terms and conditions of the following *eGenix.com Commercial License Agreement* and the terms and conditions of the third-party licenses listed in section 11.2 Third-Party Licenses.

11.1 eGenix.com Commercial License Agreement

EGENIX.COM COMMERCIAL LICENSE AGREEMENT

Version 1.3.0

1. Introduction

This "License Agreement" is between eGenix.com Software, Skills and Services GmbH ("eGenix.com"), having an office at Pastor-Loeh-Str. 48, D-40764 Langenfeld, Germany, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Terms and Definitions

The "Software" covered under this License Agreement includes without limitation, all object code, source code, help files, publications, documentation and other programs, products or tools that are included in the official "Software Distribution" available from eGenix.com.

The "Proof of Authorization" for the Software is a written and signed notice from eGenix.com providing evidence of the extent of authorizations the Licensee has acquired to use the Software and of Licensee's eligibility for future upgrade program prices (if announced) and potential special or promotional opportunities. As such, the Proof of Authorization becomes part of this License Agreement.

Installation of the Software ("Installation") refers to the process of unpacking or copying the files included in the Software Distribution to an Installation Target.

"Installation Target" refers to the target of an installation operation. Targets are defined, among other parameters, in terms of the following definitions:

1) "CPU" refers to a central processing unit which is able to store and/or execute the Software (a server, personal computer, virtual machine, or other computer-like device) using at most two (2) processors,
2) "Site" refers to a single site of a company,
3) "Corporate" refers to an unlimited number of sites of the company,
4) "Developer CPU" refers to a single CPU used by at most one (1) developer.

Additional terms may be defined as part of the Proof of Authorization.

When installing the Software on a server CPU for use by other CPUs in a network, Licensee must obtain a License for the server CPU and for all client CPUs attached to the network which will make use of the Software by copying the Software in binary or source form from the server into their CPU memory. If a CPU makes use of more than two (2) processors, Licensee must obtain additional CPU licenses to cover the total number of installed processors. The number of cores per processor does not count towards this license limitation. Virtual machines always count as one (1) CPU. If a Developer CPU is used by more than one developer, Licensee must obtain additional Developer CPU licenses to cover the total number of developers using the CPU.

"Commercial Environment" refers to any application environment which is aimed at directly or indirectly generating profit. This includes, without limitation, for-profit organizations, private educational institutions, work as independent contractor, consultant and other profit generating relationships with organizations or individuals. Governments and related agencies or organizations are also regarded as being Commercial Environments.

"Non-Commercial Environments" are all those application environments which do not directly or indirectly generate profit. Public educational institutions and officially acknowledged private non-profit organizations are regarded as being Non-Commercial Environments in the aforementioned sense.

"Educational Environments" are all those application environments which directly aim at educating children, pupils or students. This includes, without limitation, class room installations and student server installations which are intended to be used by students for educational purposes. Installations aimed at administrational or organizational purposes are not regarded as Educational Environment.

3.   License Grant

Subject to the terms and conditions of this License Agreement, eGenix.com hereby grants Licensee a non-exclusive, world-wide license to

1) use the Software to the extent of authorizations Licensee has acquired and
2) distribute, make and install copies to support the level of use authorized, providing Licensee reproduces this License Agreement and any other legends of ownership on each copy, or partial copy, of the Software.

If Licensee acquires this Software as a program upgrade, Licensee's authorization to use the Software from which Licensee upgraded is terminated.

Licensee will ensure that anyone who uses the Software does so only in compliance with the terms of this License Agreement.

Licensee may not

1) use, copy, install, compile, modify, or distribute the Software except as provided in this License Agreement;
2) reverse assemble, reverse engineer, reverse compile, or otherwise translate the Software except as specifically permitted by law without the possibility of contractual waiver; or
3) rent, sublicense or lease the Software.

4. Authorizations

The extent of authorization depends on the ownership of a Proof of Authorization for the Software.

Usage of the Software for any other purpose not explicitly covered by this License Agreement or granted by the Proof of Authorization is not permitted and requires the written prior permission from eGenix.com.

5. Modifications

Software modifications may only be distributed in form of patches to the original files contained in the Software Distribution.

The patches must be accompanied by a legend of origin and ownership and a visible message stating that the patches are not original Software delivered by eGenix.com, nor that eGenix.com can be held liable for possible damages related directly or indirectly to the patches if they are applied to the Software.

6. Experimental Code or Features

The Software may include components containing experimental code or features which may be modified substantially before becoming generally available.

These experimental components or features may not be at the level of performance or compatibility of generally available eGenix.com products. eGenix.com does not guarantee that any of the experimental components or features contained in the eGenix.com will ever be made generally available.

7.   Expiration and License Control Devices

Components of the Software may contain disabling or license control devices that will prevent them from being used after the expiration of a period of time or on Installation Targets for which no license was obtained.

Licensee will not tamper with these disabling devices or the components. Licensee will take precautions to avoid any loss of data that might result when the components can no longer be used.

8.   NO WARRANTY

eGenix.com is making the Software available to Licensee on an "AS IS" basis. SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, EGENIX.COM MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, EGENIX.COM MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

9.   LIMITATION OF LIABILITY

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL EGENIX.COM BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR (I) ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF; OR (II) ANY AMOUNTS IN EXCESS OF THE AGGREGATE AMOUNTS PAID TO EGENIX.COM UNDER THIS LICENSE AGREEMENT DURING THE TWELVE (12) MONTH PERIOD PRECEEDING THE DATE THE CAUSE OF ACTION AROSE.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO LICENSEE.

10.   Termination

This License Agreement will automatically terminate upon a material breach of its terms and conditions if not cured within thirty (30) days of written

notice by eGenix.com. Upon termination, Licensee shall discontinue use and remove all installed copies of the Software.

11. Indemnification

Licensee hereby agrees to indemnify eGenix.com against and hold harmless eGenix.com from any claims, lawsuits or other losses that arise out of Licensee's breach of any provision of this License Agreement.

12. Third Party Rights

Any software or documentation in source or binary form provided along with the Software that is associated with a separate license agreement is licensed to Licensee under the terms of that license agreement. This License Agreement does not apply to those portions of the Software. Copies of the third party licenses are included in the Software Distribution.

13. High Risk Activities

The Software is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software, or any software, tool, process, or service that was developed using the Software, could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities").

Accordingly, eGenix.com specifically disclaims any express or implied warranty of fitness for High Risk Activities.

Licensee agree that eGenix.com will not be liable for any claims or damages arising from the use of the Software, or any software, tool, process, or service that was developed using the Software, in such applications.

14. General

Nothing in this License Agreement affects any statutory rights of consumers that cannot be waived or limited by contract.

Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between eGenix.com and Licensee.

If any provision of this License Agreement shall be unlawful, void, or for any reason unenforceable, such provision shall be modified to the extent necessary to render it enforceable without losing its intent, or, if no such modification is possible, be severed from this License Agreement and shall not affect the validity and enforceability of the remaining provisions of this License Agreement.

This License Agreement shall be governed by and interpreted in all respects by the law of Germany, excluding conflict of law provisions. It shall not be governed by the United Nations Convention on Contracts for International Sale of Goods.

This License Agreement does not grant permission to use eGenix.com trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party.

The controlling language of this License Agreement is English. If Licensee has received a translation into another language, it has been provided for Licensee's convenience only.

15.   Agreement

By downloading, copying, installing or otherwise using the Software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

For question regarding this License Agreement, please write to:

eGenix.com Software, Skills and Services GmbH

Pastor-Loeh-Str. 48

D-40764 Langenfeld

Germany

EGENIX.COM PROOF OF AUTHORIZATION

1 CPU License (Example)

This is an example of a "Proof of Authorization" for a 1 CPU License. These proofs are either wet-signed by the eGenix.com staff or digitally PGP-signed using an official eGenix.com PGP-key.

1. License Grant

eGenix.com Software, Skills and Services GmbH ("eGenix.com"), having an office at Pastor-Loeh-Str. 48, D-40764 Langenfeld, Germany, hereby grants the Individual or Organization ("Licensee")

    Licensee: \<name of the licensee\>

a non-exclusive, world-wide license to use the software listed below in source or binary form and its associated documentation ("the Software") under the terms and conditions of this License Agreement and to the extent authorized by this Proof of Authorization.

2. Covered Software

    Software Name: \<product name\>

    Software Version: \<product version\>

                            (including all patch level releases)

    Software Distribution: As officially made available by

                            eGenix.com on *http://www.egenix.com/*

    Operating System: any compatible operating system

3. Authorizations

eGenix.com hereby authorizes Licensee to copy, install, compile, modify and use the Software on the following Installation Targets under the terms of this License Agreement.

    Installation Targets: one (1) CPU

Use of the Software for any other purpose or redistribution IS NOT PERMITTED BY THIS PROOF OF AUTHORIZATION.

4.   **Proof**

This Proof of Authorization was issued by

<name>, <title>

Langenfeld, <date>

Proof of Authorization Key:

<license key>

EGENIX.COM PROOF OF AUTHORIZATION

1 Developer CPU License (Example)

*This is an example of a "Proof of Authorization" for a 1 Developer CPU
License. These proofs are either wet-signed by the eGenix.com staff or
digitally PGP-signed using an official eGenix.com PGP-key.*

5.   License Grant

eGenix.com Software, Skills and Services GmbH ("eGenix.com"), having an
office at Pastor-Loeh-Str. 48, D-40764 Langenfeld, Germany, hereby grants
the Individual or Organization ("Licensee")

    Licensee: <name of the licensee>

a non-exclusive, world-wide license to use the software listed below in
source or binary form and its associated documentation ("the Software")
under the terms and conditions of this License Agreement and to the extent
authorized by this Proof of Authorization.

6.   Covered Software

    Software Name: <product name>

    Software Version: <product version>

                       (including all patch level releases)

    Software Distribution: As officially made available by

                       eGenix.com on *http://www.egenix.com/*

    Operating System: any compatible operating system

7.   Authorizations

7.1   Application Development

eGenix.com hereby authorizes Licensee to copy, install, compile, modify
and use the Software on the following Developer Installation Targets for the
purpose of developing products using the Software as integral part.

Developer Installation Targets: one (1) Developer CPU

7.2   Redistribution

eGenix.com hereby authorizes Licensee to redistribute the Software bundled with a product developed by Licensee on the Developer Installation Targets ("the Product") subject to the terms and conditions of this License Agreement for installation and use in combination with the Product on the following Redistribution Installation Targets, provided that:

1. Licensee shall not and shall not permit or assist any third party to sell or distribute the Software as a separate product;

2. Licensee shall not and shall not permit any third party to

    i. market, sell or distribute the Software to any end user except subject to the terms and conditions of this License Agreement,

    ii. rent, sell, lease or otherwise transfer the Software or any part thereof or use it for the benefit of any third party,

    iii. use the Software outside the Product or for any other purpose not expressly licensed hereunder;

3. the Product does not provide functions or capabilities similar to those of the Software itself, i.e. the Product does not introduce commercial competition for the Software as sold by eGenix.com;

4. Licensee has obtained Developer CPU Licenses for all developers and CPUs used in developing the Product.

Redistribution Installation Targets:

any number of CPUs capable of running the Product and the Software

8.   Proof

This Proof of Authorization was issued by

<name>, <title>

Langenfeld, <date>

Proof of Authorization Key:

<license key>

11.2 Third-Party Licenses

eGenix.com mxODBC Connect Server contains the following open-source third-party software components:

- **Python** - Object Oriented Programming Language

- **pyOpenSSL** - Python Interface to OpenSSL

- **OpenSSL** - Secure Socket Layer (SSL) Implementation

On Windows, the mxODBC Connect Server also uses:

- **pywin32** - Python for Windows extensions

- **Microsoft Visual C++ 9.0 Runtime DLLs** - These may only be used with the mxODBC Connect Server installation.

- **Silk Icon Set** - Icons used for the application on Windows

eGenix.com mxODBC Connect Client can **optionally** use and/or include following third party software components:

- **pyOpenSSL** - Python Interface to OpenSSL

- **OpenSSL** - Secure Socket Layer (SSL) Implementation

For copyrights, notices and license texts please see the *eGenix.com Third-Party Licenses 2.0* document which is included in the product documentation directory and also available from the *eGenix.com web-site*.