eGenix.com

mxODBC Django Database Engine

ODBC Database Interface
for the Django Web Framework

Version 1.2

Contents

1. Introduction

mxODBC has proven to be the most stable and versatile ODBC interface available for Python. It has been in production use since 1997 and is actively maintained by eGenix.com to meet the requirements of modern database applications which our customers have built on top of mxODBC.

Django is a high-level Python Web framework that encourages rapid development with a clean and pragmatic design.

mxODBC Django Database Engine is a database adapter specifically designed for Django which allows interfacing with database servers providing an ODBC interface, with direct *Django Object Relational Mapper (ORM)* integration. We currently support the ORM integration for:

- **Microsoft SQL Server:**
  Microsoft SQL Server versions 2005, 2008 and 2012

ORM support for other popular database backends will be added in upcoming releases.

Databases for which we currently do not provide ORM support can be used by directly building on top of the Python DB-API 2.0 compatible mxODBC Database API. Please see section 5.2 Direct mxODBC Database Interfacing for details.

This manual explains how to setup *mxODBC Django Database Engine* for your Django web application. It is written as a technical manual, so some knowledge of Python and the basic configuration of Django is needed.

1.1 Features

The mxODBC Django Database Engine provides the following features:

- **Fully integrated with the Django ORM:** No need to learn a new API - simply continue to use the known Django ORM interface.

- **Compatible with all current Django versions**: The mxODBC Django Database Engine supports Django 1.4 and 1.5.

- **Compatible with all recommended Python versions**: The database engine supports Python 2.6 and 2.7; both as UCS2 (narrow) and UCS4 (wide) Unicode variant on Unix platforms.

- **Full Unicode Support**: The database engine can communicate with the database using native Unicode and 8-bit encodings such as UTF-8 or CP1252.

- **Full 64-bit Support**: The underlying mxODBC 3.2 library fully supports 64-bit platforms such as Mac OS X 10.6 (Snow Leopard) and 64-bit Linux systems that use unixODBC, iODBC or DataDirect ODBC managers.

- **Cross-platform Connection Objects**: The database engine will automatically choose the right platform specific ODBC manager for you.

- **Per Connection Adjustable ODBC Manager Interface**: mxODBC supports many different ODBC managers. The mxODBC Django Database Engine allows you to select the ODBC manager on a per-connection basis.

- **Per Connection Customization of Interface Parameters**: The database engines allows adjusting many different parameters to adapt the engine to your specific database needs, should you have special requirements.

MS SQL Server Features

- **MS SQL Server fully integrated into the Django ORM**: Access MS SQL Server through the Django ORM, just like any other Django ORM database.

- **MS SQL Server Regular Expression Emulation**: Even though MS SQL Server itself does not support regular expressions, the mxODBC Django Database Engine provides an emulation for simple regular expressions to simplify porting existing applications to a SQL Server backend.

- **MS SQL Server Aggregate Function Support**: We provide a special aggregate function implementation to have the Django ORM support SQL Server aggregate functions.

- **MS SQL Server Timestamp Support**: SQL Server support millisecond accuracy on timestamps. The database engine will take care of applying the necessary rounding for the microsecond precision Python timestamps in a seamless way.

- **Support for all popular SQL Server ODBC drivers**: The mxODBC Django Database Engine supports the MS SQL Server Native Client on Windows, the MS SQL Server Native Client for Linux, as well as the FreeTDS ODBC driver. Commercial drivers from well-known driver vendors are also supported.

- **Support for accessing SQL Server from Windows and Unix platforms**: On Windows and Linux you can use the SQL Server Native Client, on other Unix platforms and Mac OS X, the FreeTDS or commercial drivers can be used.

Direct mxODBC Interface to other Databases

- **Access IBM DB2, Sybase ASE, Oracle, Teradata, Netezza, etc. directly through the proven mxODBC Database API:** Import, query and save data directly to the databases using a simple to use API and all available database specific SQL dialects, capabilities and features, including ones for which the Django ORM does not provide support.

- **Fully Python DB-API 2.0 compatible interface:** mxODBC support the Python DB-API 2.0, including many standard extensions and the full set of ODBC catalog methods for database introspection.

- **Full transaction control:** When using the mxODBC Database API, you get full control over the database connections, including opening them on demand, in auto-commit mode, dynamically scaling the number of connections per request, etc.

1.2 Django ORM Integration

The mxODBC Django Database Engine package provides ORM integration subpackages for each database backend.

In version 1.2 of the mxODBC Django Database Engine we provide support for Microsoft SQL Server 2005, 2008 and 2012 through the subpackage `mxodbc_django.ms_sql_server`. For later versions of the product, we plan to add subpackages for other database backends such as Sybase, IBM DB2 and Oracle as well.

The mxODBC Django Database Engine can be used just like most other database backends shipped with Django. We have added some additional options and features that allow adjusting the engine to your specific needs, but the defaults should work for most installations.

Configuring Django to use one of the available mxODBC Django Database Engine ORM subpackages only requires changes to your application specific settings.py file.

Note:
Some features of the Django ORM may not be supported on all the database backends due to differences between the supported set of data types and operators of the underlying database servers or database server versions.

1.3 Supported Django Versions

The Django project is a quickly evolving web framework. eGenix generally tries to keep up to date with the latest supported releases of Django, but since new Django versions often introduce subtle differences in the internal Django APIs that the mxODBC Django Database Engine has to interface to, upwards compatibility is not always guaranteed.

Version 1.2 of the mxODBC Django Database Engine supports these Django versions:

- Django 1.4
- Django 1.5

It may also work with older Django versions, as far back as Django 1.2, but those versions are no longer supported by eGenix. Support for future Django version depends on the way Django is developed. In the past, changes to the Django APIs have often made it necessary to adapt the mxODBC Django Database Engine to the new versions, so you can not expect the package to be forward compatible.

1.4 Requirements

mxODBC Django Database Engine needs these environment on Windows, Unix or Mac OS X for successful installation:

Windows

- All 32-bit Windows platforms starting with Windows 2000 are supported. 64-bit versions Windows Vista x64 and later are supported as well.
- Django 1.4 or 1.5 needs to be installed and working.
- Python 2.6 or 2.7 needs to be installed and working. You normally have one of these Python versions already installed if you are using Django 1.4 or later.
- The Windows version of the mxODBC Django Database Engine uses the Windows ODBC manager as ODBC manager, so you have to configure your ODBC data sources using its GUI interface which is available through the system settings folder.
- You should setup at least one configured and running ODBC data source for testing purposes.

Unix

- SuSE, RedHat or Ubuntu Linux distributions, as well as FreeBSD, for x86 and x86_64 processors are supported Unix platforms. We can also provide ports and custom builds for other Unix platforms such as IBM AIX or Oracle Solaris on request. Please write to *sales@egenix.com* for details.
- Django 1.4 or 1.5 needs to be installed and working.
- Python 2.6 or 2.7 needs to be installed and working. You normally have one of these Python versions already installed if you are using Django 1.4 or later.
- On Linux and FreeBSD, the binary package includes support for the unixODBC and iODBC ODBC managers. On Linux, the DataDirect ODBC manager is also supported. You must have at least one of these installed in order to be able to connect to ODBC data sources. Please use the ODBC manager GUI interfaces to configure the data sources. The Django Database Engine prefers unixODBC over iODBC over DataDirect, if more than one ODBC manager is installed.
- You should setup at least one configured and running ODBC data source for testing purposes.

Mac OS X

- Mac OS X 10.4/10.5 Intel and PPC 32-bit and Mac OS X 10.6 Intel 64-bit are supported. Please note that we are phasing out support for Mac OS X 10.4 and 10.5 as well as the 32-bit Intel version.

- Django 1.4 or 1.5 needs to be installed and working.

- Python 2.6 or 2.7 needs to be installed and working. You normally have one of these Python versions already installed if you are using Django 1.4 or later.

- Mac OS X uses a variant of iODBC as system ODBC manager. On Mac OS X 10.4 and 10.5 this comes pre-installed with the system. On Mac OS X 10.6 and later, the ODBC manager is available from Apple as *separate download*. Alternatively, you can use the new *ODBC Manager* which is maintained by *Actual Technologies*. Please use the ODBC manager GUI interfaces to configure the data sources.

- You should setup at least one configured and running ODBC data source for testing purposes.

2. Installation

The mxODBC Django Database Engine package is distributed as a third party add-on for the Django Web framework.

It comes with all components needed to enable ODBC driver access for Django:

- *egenix-mx-base*, providing the base functionality
- *egenix-mxodbc*, providing the ODBC level interface
- Django-compatible ORM database engines for the supported backends

The product does not include ODBC drivers for the database. You can get these from your database vendor or one of the many third-party vendors providing ODBC drivers for many combinations of platform and database version. See the *ODBC vendor list on SQLSummit.com* for details.

The following sub-sections will guide you through the download and installation process.

IMPORTANT NOTES:

You will have to install a **valid mxODBC Django Database Engine license** in order to run your Django application with the backends provided by mxODBC Django Database Engine.

Please do not install the eGenix packages **egenix-mx-base** or **egenix-mxodbc** separately when using the mxODBC Django Database Engine product. It already includes these packages. A separate installation is not needed and will cause conflicts.

2.1 Windows Installation

On Windows, you typically install Python using the Python MSI Installer, which places the installation details into the Windows Registry and allows the installation to be managed using the software installation manager built into Windows.

2.1.1 Installation managed by Windows

If you have a setup like this and would like to install the mxODBC Django Database Engine in the same way, please proceed as described in

- Section 2.7 Installation using Windows MSI Installer

2.1.2 Installation managed by Python

Alternatively to the Windows Installer-based installation, it is also possible to install the mxODBC Django Database Engine using **zc.builout**, **setuptools/distribute/easy_install** or **pip**. These installation mechanism are not managed by the Windows software installation manager, so cannot be uninstalled using the Windows manager. However, they offer ways of automating the installation and uninstallation from within Python.

If you'd like to use one of the mentioned methods for installation, please proceed as described in following sections:

- Section 2.4 Installation using zc.buildout
- Section 2.5 Installation using easy_install
- Section 2.6 Installation using pip

2.2 Unix / Mac OS X Installation

On Unix platforms such as Linux, FreeBSD, Mac OS X, you have to use Python installation managers to install the mxODBC Django Database Engine.

We support **zc.builout**, **setuptools/distribute/easy_install** or **pip**.

Depending on your preferred method, please proceed as described in following sections:

- Section 2.4 Installation using zc.buildout
- Section 2.5 Installation using easy_install
- Section 2.6 Installation using pip

2.3 Download the Software

eGenix distributes the mxODBC Django Database Engine in form of **.egg** and **.prebuilt files.** Both types of files are a Python binary distribution format, which allows distribution of compiled Python packages without the need for a compile step on the target machine.

While .egg files work well with setuptools/distribute based **easy_install** and **zc.buildout**, the .prebuilt files are usable with the more modern **pip** installer.

There are two ways to get the database engine installed in your Django project: an automatic approach and a manual one.

2.3.1 Automatic download

The mxODBC Django Database Engine is normally distributed and installed in form of Python egg archives which are built for automatic download and made available through a special package index on the eGenix.com website.

easy_install and zc.buildout (setuptools/distribute-based)

A separate **manual download is normally not needed**, since *easy_install* and *zc.buildout* installation and build tools will automatically find and download the .egg software archives from the eGenix.com website as needed.

pip

Please note that pip does not support automatically downloading .prebuilt files. You have to use the manual download option, if you intend to use pip.

2.3.2 Manual download

If you do need to download the egg or prebuilt archives eGenix makes available, e.g. because you want to use pip, your server doesn't have Internet access, or is behind a firewall, please read on.

You can download the binary egg and prebuilt archives for your combination of platform, Python version and Unicode variant from the eGenix.com web-site at *http://www.egenix.com/*.

Choosing the right file to download

Please make sure that you download the right version for your Django installation. If you get import errors, notices of failed initialization or Django hangs, you likely have the wrong product version installed.

These parameters make a difference:

Installation Tool (easy_install/zc.buildout or pip)

Depending on which installation tool you will be using to install the database engine in your Django project, you will have to download either an .egg or a .prebuilt archive version:

- download **.egg files** if you are using **easy_install** and **zc.buildout**
- download **.prebuilt** files if you are using **pip**
- download **.msi** files if you are using the **Windows Installer**

Platform (Windows, Linux, FreeBSD, Mac OS X)

All recent versions of these operating systems are supported. Just be sure to download the correct archive for your platform.

Python Build Version (2.6, 2.7)

To check which Python version your Django installation is using, startup the Python interpreter[1] using the –V option:

```
bin/python –V
```

This will print out the Python version number.

Python Build Architecture (32 bit or 64 bit)

On many platforms we support x86 32-bit and x86_64 64-bit versions of Python.

To find out which version Django is using, run the following command:

```
bin/python -c "import struct; print struct.calcsize('P')*8,'bit'"
```

This will print out "32 bit" or "64 bit".

Unicode Variant (UCS2 or UCS4)

On Windows, Python is always compiled as UCS2 version, so you can skip this section.

On Unix, Python can be built using two different Unicode variants: UCS2 and UCS4.

To find out which variant your Python version was compiled with, run the following command (if you are running Django with a different Python interpreter, please replace bin/python with the one you are using):

```
bin/python -c "print('UCS%s'%len(u'x'.encode('unicode-internal')))"
```

This will either print out "UCS2" or "UCS4".

Most Django installation will be using the system's default Python installation, either directly or via a *virtualenv* local installation.

On Linux and FreeBSD this usually means you have a UCS4 version of Python.

[1] Have a look at the ./bin/django-admin.py startup file in your Django directory to find the path to the Python interpreter.

2.4 Installation using zc.buildout

This section explains the installation of the mxODBC Django Database Engine using a *zc.buildout* based approach. zc.buildout is a software configuration and setup tool that allows for a programmatic, repeatable approach to software deployment.

2.4.1 Before You Start

The binary installation archives and egg files include everything you need to run the mxODBC Django Database Engine, including the necessary *egenix-mx-base* and *egenix-mxodbc* packages for Django.

> Please make sure that you **do not have egenix-mx-base or egenix-mxodbc installed separately**, since the installation will not succeed in such a setup.
>
> If you have not installed them manually in your Python installation, also **make sure that you don't have any of these buildout recipes installed**: `collective.recipe.mxodbc` or `collective.recipe.mxbase`.

Upgrading

zc.buildout will automatically upgrade your mxODBC Django Database Engine to the latest release.

If you don't want this to happen, add an entry with the exact version number to the `[versions]` section of the buildout.cfg or versions.cfg file, e.g.

```
[versions]
egenix-mxodbc-django = 1.2.0
```

License Files

In order to run the mxODBC Django Database Engine, you will need license files from eGenix.com.

If you want to test the product before buying it, you can request evaluation licenses via the eGenix.com web-site at *http://www.egenix.com/*.

When buying licenses from the eGenix.com online shop (*http://shop.egenix.com/*), you will receive the license files immediately after purchase.

In both cases, the license files are sent to the email address you specified during the purchase process or from which you wrote the evaluation license request in form of a ZIP license archive attached to the license email – usually named licenses.zip.

The license archive licenses.zip contains one subdirectory per Django Project license you bought. The directories are named after the license key for each Django Project license. A typical license archive will have these contents:

```
2100-8789-0322-0926-2568-6429/mxodbc_django_license.py
2100-8789-0322-0926-2568-6429/mxodbc_django_license.txt
2100-8089-0312-0926-2668-6529/mxodbc_django_license.py
2100-8089-0312-0926-2668-6529/mxodbc_django_license.txt
```

(in the above example, the license archive contains the files for two product licenses).

In order to install the license files, please unzip the license archive to your Django installation directory, i.e. the directory with the buildout.cfg file. zc.buildout will need to find the files for proper operation.

> The files will be copied to the right Django Project directory location via a zc.buildout recipe where Django can find them, so **you should not remove these license directories.**

2.4.2 Step-by-step Installation Guide

We assume that you have already installed Django and unzipped the license files to the installation directory as explained in the previous section.

Step 1

Determine whether you are using a UCS2 or UCS4 build of Python.

> **Windows users** always need the UCS2 version. **Mac OS X users** will most likely also need the UCS2 version, since Python's default configuration is to build a UCS2 interpreter. **Linux users** will likely need a UCS4 build.

To find out which variant your Python version was compiled with, run the following command (if you are running Django with a different Python interpreter, please replace bin/python with the one you are using):

```
bin/python -c "print('UCS%s'%len(u'x'.encode('unicode-internal')))"
```

This will either print out “UCS2” or “UCS4”.

Step 2

We assume that you are using a buildout recipe similar to the *djangorecipe* recipe, which has a `[django]` section for installing Django and the Django project.

In your buildout.cfg file, please add/adapt the following content:

```
[buildout]
…

### Add eGenix Index to the buildout setup
#
# IMPORTANT: Use the URL
#     https://downloads.egenix.com/python/index/ucs2/
# if your Python version is a UCS build. If you have a UCS4
# build of Python, use the URL
#     https://downloads.egenix.com/python/index/ucs4/
#
find-links =
    …
    https://downloads.egenix.com/python/index/ucs2/

### Add eGenix mxODBC Django Database Engine eggs
#
# The new egenix-mxodbc-django-license part takes care of
# automatically installing your license files in the instance.
#
eggs =
    …
    egenix-mxodbc-django

parts =
    …
    egenix-mxodbc-django-license

…

### Install the Django Project licenses for egenix-mxodbc-django
#
# This part copies the license files you extracted to the
# project directory to the directory where your project's
# settings.py module is located.
#
# IMPORTANT: You need to replace ***license-serial*** with the
# directory containing the mxodbc_django_license.py file for
# your project.
#
[egenix-mxodbc-django-license]
recipe = collective.recipe.template
input = ***license-serial***/mxodbc_django_license.py
output =
     ${django:project}/${django:project}/mxodbc_django_license.py

### Define versions of packages to be used
#
# buildout will automatically use the latest version it finds
# for building instances. This may not always be what you
# want, so it's usually better to pin down the version you're
# interested in.
#
[versions]
…
```

```
egenix-mxodbc-django = 1.2.0
collective.recipe.template = 1.9
```

In the above file, you have to make two adjustments:

- Adjust the **URL used in the find-links directive** to use either the ucs2/ or the ucs4/ version of the eGenix PyPI-style distribution index.
- Adjust the **egenix-mxodbc-django version** to the one that you would like to use.
- Replace the *****license-serial***/ path component** with the license directory containing the license for the instance you are configuring. The directories extracted from the license.zip file are usually named after the license serial, e.g. 2100-8789-0322-0926-2568-6429/.

Step 3

Run buildout in the installation directory:

```
./bin/buildout
```

This will rebuild your Django project using the newly added eGenix mxODBC Django Database Engine product.

Step 4

To complete the installation, configure a database connection to use the mxODBC Django Database Engine and restart Django.

Here's a quick example of a settings.py entry which uses the MS SQL Server package of the database engine:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {},
    }
}
```

The configuration of the mxODBC Django Database Engine is explained in more detail further below.

2.5 Installation using easy_install

This section explains the installation of the mxODBC Django Database Engine using an easy_install based approach. easy_install is a script that is installed as part of the *distribute* or *setuptools* Python packaging system.

It uses egg files which contain binary Python packages for easy installation.

2.5.1 Before You Start

The binary installation egg files include everything you need to run the mxODBC Django Database Engine, including the necessary *egenix-mx-base* and *egenix-mxodbc* packages for Django.

Please make sure that you **do not have egenix-mx-base or egenix-mxodbc installed separately**, since the installation will not succeed in such a setup.

Upgrading

easy_install will automatically upgrade your mxODBC Django Database Engine to the latest release, if you run it with option --upgrade.

If want to install or upgrade a specific release, please specify the version as requirement, e.g. `easy_install egenix-mxodbc-django==1.2.0`.

License Files

In order to run the mxODBC Django Database Engine, you will need license files from eGenix.com.

If you want to test the product before buying it, you can request evaluation licenses via the eGenix.com web-site at *http://www.egenix.com/*.

When buying licenses from the eGenix.com online shop (*http://shop.egenix.com/*), you will receive the license files immediately after purchase.

In both cases, the license files are sent to the email address you specified during the purchase process or from which you wrote the evaluation license request in form of a ZIP license archive attached to the license email – usually named licenses.zip.

The license archive licenses.zip contains one subdirectory per Django Project license you bought. The directories are named after the license key for each Django Project license. A typical license archive will have these contents:

```
2100-8789-0322-0926-2568-6429/mxodbc_django_license.py
2100-8789-0322-0926-2568-6429/mxodbc_django_license.txt
2100-8089-0312-0926-2668-6529/mxodbc_django_license.py
2100-8089-0312-0926-2668-6529/mxodbc_django_license.txt
```

(in the above example, the license archive contains the files for two product licenses).

In order to install the license files, please unzip the license archive to your Django Project and place the `mxodbc_django_license.*` files into the directory where your project's `settings.py` Django module is located.

If the mxODBC Django Database Engine cannot find the license module on import, Django will give an error on startup.

2.5.2 Step-by-step Installation Guide

We assume that you have already installed Django and unzipped the license files to the project directory as explained in the previous section.

Step 1

Determine whether you are using a UCS2 or UCS4 build of Python.

Windows users always need the UCS2 version. **Mac OS X users** will most likely also need the UCS2 version, since Python's default configuration is to build a UCS2 interpreter. **Linux users** will likely need a UCS4 build.

To find out which variant your Python version was compiled with, run the following command (if you are running Django with a different Python interpreter, please replace bin/python with the one you are using):

```
bin/python -c "print('UCS%s'%len(u'x'.encode('unicode-internal')))"
```

This will either print out “UCS2” or “UCS4”.

Step 2

You are now ready to install the eGenix Django Database Engine using easy_install. Select one of the following variants depending on the outcome of the UCS-test in the previous step.

If you are using a **UCS2 version of Python**, please run the following command using the `easy_install` script corresponding to your Python installation (usually in the same directory as the `python` binary).

```
easy_install \
   -i https://downloads.egenix.com/python/index/ucs2/ \
   egenix-mxodbc-django
```

(please use the correct index URL for your Python version)

For a **UCS4 version of Python**, run:

```
easy_install \
   -i https://downloads.egenix.com/python/index/ucs4/ \
   egenix-mxodbc-django
```

The above commands install the latest available version of the egenix-mxodbc-django package. If you'd like to **install a specific version**, please add a version restriction ("`egenix-mxodbc-django==1.2.0`"), e.g.

```
easy_install \
   -i https://downloads.egenix.com/python/index/ucs4/ \
   egenix-mxodbc-django==1.2.0
```

This will install version 1.2.0 of the mxODBC Django Database Engine.

Step 3

To complete the installation, configure a database connection to use the mxODBC Django Database Engine and restart Django.

Here's a quick example of a settings.py entry which uses the MS SQL Server package of the database engine:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {},
    }
}
```

The configuration of the mxODBC Django Database Engine is explained in more detail further below.

2.6 Installation using pip

This section explains the installation of the mxODBC Django Database Engine using the *pip package manager*.

pip uses a standard setup.py based approach for installation. We have created a special prebuilt binary format which supports this kind of interface and works well with pip.

Please note that you have to download the .prebuilt package file from our website to install the egenix-mxodbc-django package, since pip can not automatically select the correct file for your installation platform.

2.6.1 Before You Start

The binary installation prebuilt files include everything you need to run the mxODBC Django Database Engine, including the necessary *egenix-mx-base* and *egenix-mxodbc* packages for Django.

> Please make sure that you **do not have egenix-mx-base or egenix-mxodbc installed separately**, since the installation will not succeed in such a setup.

Upgrading

To force an upgrade, please add the --upgrade option to the pip installation command below.

Upgrading to a specific version is simply done by downloading and using the specific .prebuilt archive version for the installation.

License Files

In order to run the mxODBC Django Database Engine, you will need license files from eGenix.com.

If you want to test the product before buying it, you can request evaluation licenses via the eGenix.com web-site at *http://www.egenix.com/*.

When buying licenses from the eGenix.com online shop (*http://shop.egenix.com/*), you will receive the license files immediately after purchase.

In both cases, the license files are sent to the email address you specified during the purchase process or from which you wrote the evaluation license request in form of a ZIP license archive attached to the license email – usually named licenses.zip.

The license archive licenses.zip contains one subdirectory per Django Project license you bought. The directories are named after the license key for each Django Project license. A typical license archive will have these contents:

```
2100-8789-0322-0926-2568-6429/mxodbc_django_license.py
2100-8789-0322-0926-2568-6429/mxodbc_django_license.txt
2100-8089-0312-0926-2668-6529/mxodbc_django_license.py
2100-8089-0312-0926-2668-6529/mxodbc_django_license.txt
```

(in the above example, the license archive contains the files for two product licenses).

In order to install the license files, please unzip the license archive to your Django Project and place the `mxodbc_django_license.*` files into the directory where your project's `settings.py` Django module is located.

If the mxODBC Django Database Engine cannot find the license module on import, Django will give an error on startup.

2.6.2 Step-by-step Installation Guide

We assume that you have already installed Django and unzipped the license files to the project directory as explained in the previous section.

Step 1

Determine whether you are using a UCS2 or UCS4 build of Python.

> **Windows users** always need the UCS2 version. **Mac OS X users** will most likely also need the UCS2 version, since Python's default configuration is to build a UCS2 interpreter. **Linux users** will likely need a UCS4 build.

To find out which variant your Python version was compiled with, run the following command (if you are running Django with a different Python interpreter, please replace bin/python with the one you are using):

```
bin/python -c "print('UCS%s'%len(u'x'.encode('unicode-internal')))"
```

This will either print out “UCS2” or “UCS4”.

Step 2

Using the information obtained in step 1 and the name of your installation platform, download the right .prebuilt package archive from the *eGenix.com website* and place it into the Django installation directory.

Step 3

You are now ready to install the eGenix Django Database Engine using `pip`.

Please run the following command using the `pip` script corresponding to your Python installation (usually in the same directory as the `python` binary).

```
pip install egenix-mxodbc-django-<version info>-prebuilt.zip
```

(replace `egenix-mxodbc-django-<version info>-prebuilt.zip` with the filename you've downloaded)

This will install the mxODBC Django Database Engine.

Uninstallation, e.g. if you've accidentally installed a wrong version, can be done using the standard pip uninstall command:

```
pip uninstall egenix-mxodbc-django
```

Step 4

To complete the installation, configure a database connection to use the mxODBC Django Database Engine and restart Django.

Here's a quick example of a settings.py entry which uses the MS SQL Server package of the database engine:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {},
    }
}
```

The configuration of the mxODBC Django Database Engine is explained in more detail further below.

2.6.3 Uninstall

The pip installer keeps track of the files it installed for a package.

To uninstall the mxODBC Django Database Engine, run the following command:

```
pip uninstall egenix-mxodbc-django
```

This will uninstall all files that can safely be removed from the system. It will not remove files which were added to the subpackages after installation, nor will it remove the license files you manually installed.

2.7 Installation using Windows MSI Installer

This section explains the installation of the mxODBC Django Database Engine using the native Windows MSI installer files.

These installations create registry entries and can be uninstalled through the standard Windows OS Software Management tools. The MSI installers we provide allow for interactive, unattended and automated installs. Please see the *Python MSI installer features* page on python.org for details on available options.

In order to use the MSI installers, you have to download the .msi installer file from the *mxODBC Django Database Engine product page*.

2.7.1 Before You Start

The binary MSI installers include everything you need to run the mxODBC Django Database Engine, including the necessary *egenix-mx-base* and *egenix-mxodbc* packages for Django.

Please make sure that you **do not have egenix-mx-base or egenix-mxodbc installed separately**, since the installation will not succeed in such a setup.

Upgrading

When upgrading, we generally recommend uninstalling the previous installation using the Windows Software Management tools first.

After the uninstall has completed, you can then proceed as usual with the installation.

License Files

In order to run the mxODBC Django Database Engine, you will need license files from eGenix.com.

If you want to test the product before buying it, you can request evaluation licenses via the eGenix.com web-site at *http://www.egenix.com/*.

When buying licenses from the eGenix.com online shop (*http://shop.egenix.com/*), you will receive the license files immediately after purchase.

In both cases, the license files are sent to the email address you specified during the purchase process or from which you wrote the evaluation license request in form of a ZIP license archive attached to the license email – usually named licenses.zip.

The license archive licenses.zip contains one subdirectory per Django Project license you bought. The directories are named after the license key for each Django Project license. A typical license archive will have these contents:

```
2100-8789-0322-0926-2568-6429/mxodbc_django_license.py
2100-8789-0322-0926-2568-6429/mxodbc_django_license.txt
2100-8089-0312-0926-2668-6529/mxodbc_django_license.py
2100-8089-0312-0926-2668-6529/mxodbc_django_license.txt
```

(in the above example, the license archive contains the files for two product licenses).

In order to install the license files, please unzip the license archive to your Django Project and place the `mxodbc_django_license.*` files into the directory where your project's `settings.py` Django module is located.

If the mxODBC Django Database Engine cannot find the license module on import, Django will give an error on startup.

2.7.2 Step-by-step Installation Guide

We assume that you have already installed Django and unzipped the license files to the project directory as explained in the previous section.

Step 1

If you are working on a Windows x64 system, you need to determine whether you are using a 64-bit or a 32-bit build of Python on your Windows system. On Windows x86 you can skip this step, since you'll always have a 32-bit build of Python as well.

You can determine the Python variant by running the following command:

```
bin/python -c "import struct; print struct.calcsize('P')*8,'bit'"
```

The output will tell you whether your Python installation is a 32-bit or a 64-bit one.

Step 2

Using the information obtained in step 1 and the variant of your installation Windows platform (Windows x86 or x64), please download the right .msi installer archive from the *eGenix.com website* and place it into the Django installation directory.

Step 3

You are now ready to install the eGenix Django Database Engine. Simply double-click on the `egenix-mxodbc-django-….msi` file and follow the instructions. This will install the mxODBC Django Database Engine.

During the installation, you have to answer a few user access control (UAC) dialogs. Depending on the installation location of Python, it may also be necessary to run the MSI file as administrator.

> Please note that it's better to "install for all users", since per-account installations of Python tend to cause permission problems.

Step 4

To complete the installation, configure a database connection to use the mxODBC Django Database Engine and restart Django.

Here's a quick example of a settings.py entry which uses the MS SQL Server package of the database engine:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {},
    }
}
```

The configuration of the mxODBC Django Database Engine is explained in more detail further below.

2.7.3 Uninstall

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

To uninstall the mxODBC Django Database Engine, run the Windows Software Management tool and select the "*Python x.x eGenix mxODBC Django Database Engine x.x*" entry for deinstallation.

This will uninstall all files that can safely be removed from the system. It will not remove files which were added to the subpackages after installation, nor will it remove the license files you manually installed.

3. Configuration

The configuration of access to a database involves two steps:

1. Configuration of the database as ODBC data source
2. Connecting Django to the data source using the mxODBC Django Database Engine

The next sections explain the details of these two steps.

3.1 ODBC Data Source Configuration

Before being able to connect to a database, you have to configure the database as data source in the Operating System's ODBC manager.

3.1.1 General Notes

These notes apply to all platforms.

Connection Pooling by the ODBC Manager

As of version 1.5, Django does not provide database connection pooling. This can result in poor performance, since Django usually reconnects to the database on every single request.

Fortunately, ODBC manager usually come with connection pooling built-in, so it's possible to work around this problem.

You can turn on connection pooling in your ODBC manager's GUI, whether on Linux, Mac OS X or Windows system. If you'd rather like to enable the setting without using a GUI, please consult your ODBC manager's documentation.

When using ODBC manager connection pooling, please make sure that:

- your web application still works as intended
- performance does goes up

Enabling connection pooling in the ODBC manager can have unwanted side-effects, e.g. due to connection settings leaking across requests, which can result in the web application becoming unstable.

It can also result in poor performance with some databases, so testing the setting is needed and no general recommendation can be given.

3.1.2 Windows Platform

On Windows, you must configure the ODBC manager through the standard system settings dialogs (*ODBC Data Sources*).

Please consult the Windows help files and your database/ODBC driver documentation for details on how to setup data sources in the Windows ODBC Manager.

> Note that if you plan to run **Django as Windows service**, it may be necessary to setup the ODBC data sources as System-DSN. Otherwise, the Django process won't be able to see or access the ODBC data sources you setup in the Windows ODBC manager.

Platform Default ODBC Manager

The platform default ODBC manager (the one selected using `mxodbc_manager="Manager"` in the connection options) on Windows is always the Windows ODBC manager.

On 64-bit Windows platforms, Windows comes with two versions of the Windows ODBC manager: a 32-bit version and a 64-bit version. The 32-bit version of mxODBC Django Database Engine will choose the 32-bit one, the 64-bit version of mxODBC Django Database Engine the 64-bit ODBC manager.

3.1.3 Unix Platform

On Unix (Linux or Solaris), it suffices to supply a standard ODBC INI file either as /etc/odbc.ini or in the Django user home directory as ~/.odbc.ini (note the leading '.') file which uses the same syntax as the Windows file ODBC.INI.

Alternatively, you can use the unixODBC/iODBC/DataDirect management GUIs which allows setting up data sources in the same way as the Windows ODBC manager provides on Windows.

Details on the ODBC manager configuration on Unix can be found on the websites of the ODBC managers:

unixODBC - *http://www.unixodbc.org/*

iODBC - *http://www.iodbc.org/*

DataDirect - *http://www.datadirect.com/*

Please consult your database / ODBC driver documentation for details on how to setup data sources using these ODBC managers.

Note that you only need to have one of these ODBC managers installed on the installation machine for the mxODBC Django Database Engine to work.

Platform Default ODBC Manager

The platform default ODBC manager (the one selected using `mxodbc_manager="Manager"` in the connection options) depends on which ODBC manager mxODBC Django Database Engine finds during startup. It select the first one found from the above given list, i.e. unixODBC, iODBC, DataDirect.

3.1.4 Mac OS X Platform

On Mac OS X, please configure the ODBC manager through the standard system *ODBC Administrator*. Open the finder and navigate to Applications / Utilities / ODBC Administrator.

Internally, the ODBC Administrator builds upon the open-source ODBC manager **iODBC**, so the comments related to iODBC also apply to the Mac OS X ODBC manager.

If you are running Mac OS X 10.6 or later and don't have the ODBC Administrator installed, you can *download and install it from Apple*.

Please consult the Mac OS X help and your database/ODBC driver documentation for details on how to setup data sources in the Mac OS X ODBC Administrator.

> If you are running on Mac OS X 10.6 and have **problems finding the data sources** configured with the ODBC Administrator in the mxODBC Django Database Engine data source list or connecting to them, please see *this Mac Dev Center article* for a fix.

Platform Default ODBC Manager

The platform default ODBC manager (the one selected using `mxodbc_manager="Manager"` in the connection options) depends on which ODBC manager mxODBC Django Database Engine finds during startup. It select the first one found from the above given list, i.e. unixODBC, iODBC.

Unless you have installed the unixODBC ODBC manager by hand or via Mac Ports and configured an appropriate linker setup, this will select the iODBC ODBC manager as default.

3.2 ODBC Driver/Manager Troubleshooting

This section collects a few hints and tricks we have gathered during the beta testing and rollout phase which may be helpful in setting up a working ODBC connection.

Since ODBC drivers can sometimes vary in quality and features, care has to be taken when configuring the ODBC drivers so that you get the best performance and stability possible.

> Please note that some of the following sections on ODBC drivers do not may not apply to the current version of the mxODBC Django Database Engine, since the respective database backends are not supported by the Django ORM integration. We still include them, since it is well possible to connect to these databases directly via the include underlying mxODBC Python interface.

3.2.1 Windows ODBC Manager

The Windows ODBC manager implements a feature called **Connection Pooling** which allows faster connects to databases. In some cases we have observed failures and problems when using the connection pooling feature of the ODBC manager together with the mxODBC Django Database Engine.

If you are observing similar problems, we suggest that you turn off connection pooling in the Windows ODBC manager for those data sources that you wish to use the Django Database Engine for.

3.2.2 Unix ODBC Managers iODBC, unixODBC and DataDirect

On Unix the mxODBC Django Database Engine uses an already installed iODBC, unixODBC or DataDirect manager to communicate with the installed ODBC drivers.

At Django startup time, the Django Database Engine tries to import the interfaces for the ODBC managers and writes a notice to the Django startup shell window. Django can only use those interfaces which are successfully imported at this point.

If all interfaces fail to load, the mxODBC Django Database Engine will not be usable.

Typical problems which prevent the mxODBC Django Database Engine from correctly importing the underlying mxODBC interfaces to the ODBC managers are:

- missing ODBC manager installations,
- missing permissions of the Django user account to access the shared libraries of the ODBC managers (these are typically called libiodbc.so and libodbc.so),
- incorrectly setup linker parameters: the dynamic linker cannot find the shared libraries; this can usually be remedied by setting the LD_LIBRARY_PATH environment variable,
- incompatible ODBC manager versions.

If you use recent versions of the iODBC, unixODBC or DataDirect ODBC managers, the last point is less likely, since eGenix always builds the binary distributions of the mxODBC Django Database Engine against the latest stable releases of these managers.

3.2.3 Microsoft Access ODBC Driver

The MS Access database uses the Jet Engine to access the database. ODBC drivers for the Jet Engine prior to version 4.0 are *not* thread-safe and can cause problems if used with mxODBC Django Database Engine.

Please make sure that you have the latest revision of the Jet Engine and corresponding ODBC drivers installed.

> If you get an **error *HY024* mentioning an invalid option** value during connect, it is likely that the data source is an auto-commit-only data source (meaning that it doesn't support transactions), e.g. a file data source.

In such a case:

- create a connection object that is initially closed,
- go to the properties tab of the connection object,
- select "Use Auto-Commit" and "Open Connection"
- click "Save Changes"

The connection should now be opened in auto-commit mode. Note that the data source will not participate in the Django transaction mechanism. You should only use such data sources for reading data, not writing data.

3.2.4 IBM DB2 ODBC Driver

The DB2 ODBC Driver for Windows has an optimization option called "early cursor close" (or similar). This has to be switched off. Otherwise, you'll get lots of SQLSTATE 08001 or 080003 errors during connects and parallel execution of SQL Methods becomes impossible.

3.2.5 SAP DB ODBC Driver

Some versions of the SAP DB ODBC driver have a problem with reporting the correct scale of float columns.

This should not harm the functionality of the mxODBC Django Database Engine.

3.2.6 FreeTDS ODBC Driver (access MS SQL Server from Linux)

The FreeTDS ODBC driver is a free ODBC driver for Unix which allows you to connect from Unix to Sybase and/or Microsoft's SQL Server running on different platforms such as Windows 2000.

Please note that the driver's current versions (0.9x) still lack a few ODBC features which you may need for production work. Unicode support was added just recently in version 0.91.

To work around the showstopper bugs in the driver, eGenix has added a set of compatibility features to the underlying mxODBC interface to at least make the setup mxODBC Django Database Engine + FreeTDS driver usable for standard queries to the supported databases. See the *mxODBC Documentation* for hints on how to setup FreeTDS to work together with the mxODBC Django Database Engine.

For production systems, we recommend deploying professional quality ODBC drivers to access MS SQL Server and/or Sybase, such as the ones available from EasySoft, OpenLink and DataDirect.

3.2.7 MS SQL Server Native Client for Linux

This is a new ODBC driver from Microsoft which was ported from the existing mature SQL Server Native Client driver version 11 on Windows. It is currently only available for 64-bit Linux variants.

It is more robust than the FreeTDS ODBC driver and provides better Unicode support, but also has the same issues as the SQL Server Native Client driver on Windows.

Please see the *mxODBC documentation* for details on setting up the driver.

3.2.8 PostgreSQL ODBC Driver

The PostgreSQL project has an ODBC driver which is available for Windows as binary and also compiles on Unix from source.

On Unix, the driver is typically included in unixODBC ODBC manager binary packages, so you may have the driver already installed if you're running Django on Unix and have unixODBC installed (the ODBC driver file is called psqlodbc.so).

Connecting to PostgreSQL using e.g. unixODBC or the Windows ODBC manager works just like for all other databases.

The only known problem with the ODBC driver for PostgreSQL is the lack of support for BLOBs (binary long objects). Please refer to the ODBC driver documentation for ways to work-around this caveat in the driver. Apart from that data type, all basic data types are supported.

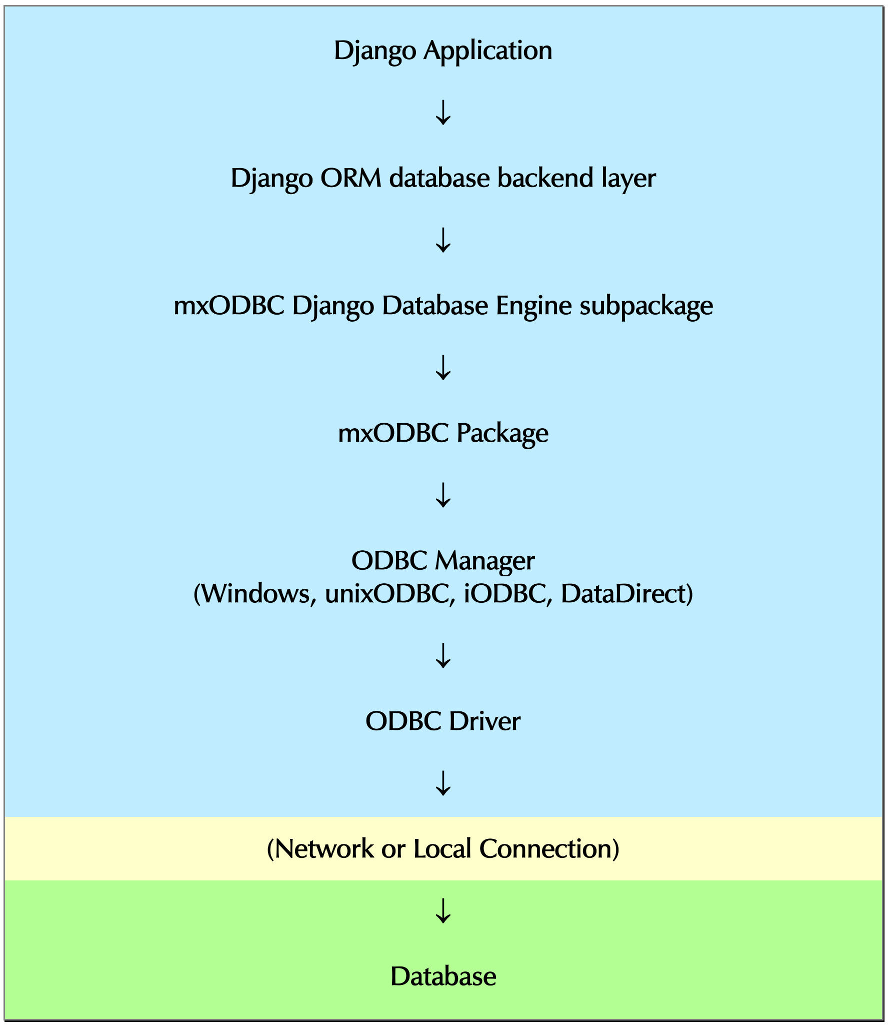
3.2.9 Other ODBC Drivers and Manager Setups

More information about various ODBC driver and manager setups can be found in the *mxODBC Documentation: Interface – Subpackages - General Notes*.

4. Setting up your Django application

mxODBC Django Database Engine provides backends can be used with any Django application to access database servers through mxODBC and the ODBC interface in general.

The typical setup looks like this:


Django Application
↓
Django ORM database backend layer
↓
mxODBC Django Database Engine subpackage
↓
mxODBC Package
↓
ODBC Manager
(Windows, unixODBC, iODBC, DataDirect)
↓
ODBC Driver
↓
(Network or Local Connection)
↓
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 possibly also on a different machine.

Please consult with the *mxODBC Manual* for further information on the configuration of data sources and ODBC drivers.

4.1 Configuring database access

You will need to make some changes to the settings.py file of your Django application. Let's start with an example configuration:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=example;UID=sa;PWD=123',
        'OPTIONS': {},
    }
}
```

The above configuration will connect to the ODBC data source (DSN) "`example`" using the UID (user name) and PWD (password) credentials given in the DSN entry of the Django database connection.

The DSN must point to a database stored in a Microsoft SQL Server instance, since we defined `ms_sql_server` as the mxODBC Django Database Engine subpackage to use.

Note that the database server does not need to be on the same machine as your Django application, but you have to make sure that the ODBC driver can connect to the server.

There are also a few backend options supported by mxODBC Django Database Engine to configure details specific to mxODBC or the database server in use. Please see the next sections for details.

4.1.1 Database settings

mxODBC Django Database Engine supports the following database settings:

`ENGINE`

String containing the full (dotted) name of the mxODBC Django Database Engine subpackage to use.

Possible values:

`mxodbc_django.ms_sql_server`

MS SQL Server subpackage

Please make sure that you select the right subpackage for your database backend.

`DSN`

This variable must be set to the ODBC connection string for the data source you have configured in the system's ODBC manager.

The general format is:

```
DSN=<data source name>;UID=<user name>;PWD=<password>
```

The ODBC driver defined for the data source may support additional configuration options that you can specify in the connection string, e.g. the database name or a host name to connect to.

It is also possible to use the DSN string to open connections to databases that are not defined in the ODBC manager (DSN-less connections), if you know the name of the ODBC driver and other connection details such as network locations, ports, etc.

Please consult your ODBC manager/driver's documentation for details. The *mxODBC User Manual* also provides some additional advice on how to configure commonly used ODBC drivers.

Note: While it's possible to use the settings `USER` and `PASSWORD` to define the user name and password (for compatibility with other Django database engines), specifying these values in the `DSN` variable is the preferred way to setup the mxODBC Django Database Engine.

`NAME`

Django name of the database entry.

Note that the name of the database specified in the settings module has no relevance for the data source defined by the DSN setting. It is used by Django internally to identify the database and detect aliases in the DATABASES settings dictionary.

You can specify the database to be used by the database entry in the DSN connection string via the `Database=` parameter, or in the data source setup of the ODBC manager. The details depend on the used ODBC driver and manager. Please check their documentation for details. The *mxODBC User Manual* has an extensive section with examples of how to setup databases connection strings and configure ODBC drivers.

`USER`

Database user name to connect with.

You only need to specify this variable, if you have not provided the user name as `UID=<user name>` in the `DSN` variable.

`PASSWORD`

Password of the user to connect with.

You only need to specify this variable, if you have not provided the user name as `PWD=<password>` in the `DSN` variable.

`OPTIONS`

Dictionary of backend options.

See section 4.1.2 Database backend options for details.

Example:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {
            'mxodbc_encoding': 'cp1252',
        },
    }
}
```

or, using the standard Django settings for USER and PASSWORD:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008',
        'USER': 'sa',
        'PASSWORD': '123',
        'OPTIONS': {
            'mxodbc_encoding': 'cp1252',
        },
    }
}
```

4.1.2 Database backend options

mxODBC Django Database Engine supports the following backend options which can be defined as key-value pairs in the `OPTIONS` database setting dictionary:

`autocommit`

Turn on/off auto commit on the connection.

Auto commit means that changes on the database connection are immediately written to the database, even if the processed web request causes an error.

Note that turning on `autocommit` should really only be done for read-only connections. A connection running in auto commit mode can easily cause data corruption in case of errors during the processing of a request.

Default is to disable auto commit when connecting to the database. Set the option to True to enable auto commit.

`mxodbc_application_encoding`

Name of the encoding to assume for 8-bit string literals that are quoted to be added literally to SQL statements built by the Django ORM.

mxODBC Django Database Engine needs to apply this conversion, since it converts all SQL statements to Unicode prior to passing them to the database.

Defaults to '`utf-8`'.

`mxodbc_connection_encoding`

Name of the encoding used to encode/decode data passed to/from the ODBC driver. mxODBC's `connection.encoding` is set to this value. It uses the encoding to convert text data between the database and the application, in case the ODBC driver does not know how to handle Unicode or the database requests text data while Django sends Unicode.

The default value is '`cp1252`' for the FreeTDS ODBC driver versions prior to 0.91 and '`utf-8`' for all others.

Note that default is to send Unicode data as native Unicode to the ODBC driver. See the `mxodbc_stringformat` option for details.

`mxodbc_datetime_as_string`

Forces the conversion of all date, time and datetime objects into strings before passing them to the ODBC driver as a parameter.

The conversion is disabled by default.

You can enable or disable it by setting the option to True or False.

`mxodbc_exact_collation`

Name of the database server collation used for exact string comparisons.

Django's ORM expects the chosen collation to be case and accent sensitive. The sort order can be adjusted to suit your needs (binary or some dictionary order).

Defaults to '`Latin1_General_BIN`' for Microsoft SQL Server.

Please see your database backend documentation for available collation names.

`mxodbc_inexact_collation`

Name of the database server collation used for inexact string comparisons.

Django's ORM expects the chosen collation to be a case insensitive, but accent sensitive collation. The sort order can be adjusted to suit your needs (binary or some dictionary order).

Defaults to '`Latin1_General_CI_AS`' for Microsoft SQL Server.

Please see your database backend documentation for available collation names.

`mxodbc_init`

List of SQL statements to execute right after initiating the ODBC connection. This option can be used to implement database specific configurations of the connection, which are not possible through ODBC driver settings.

Each item can be an string containing an SQL statement or a `(sql, parameters)` tuple.

Defaults to an empty list.

`mxodbc_manager`

Name of the mxODBC subpackage / ODBC manager to use.

Defaults to '`Manager`', which automatically selects the appropriate ODBC Manager for the platform running the Django application.

Possible values:

`Manager`

Automatically select an appropriate ODBC manager. On Windows, this selects the Windows ODBC manager. On Unix platforms, the first available manager from the following ODBC managers is chosen: unixODBC, iODBC, DataDirect.

`Windows`

Windows ODBC manager. Only possible other option on Windows. Not available on other platforms.

`unixODBC`

unixODBC ODBC manager. Unix only.

`iODBC`

iODBC ODBC manager. Unix only.

`DataDirect`

DataDirect ODBC manager. Currently only available for Linux platforms.

`mxodbc_monkey_patch_aggregates`

Enables monkey patching the Django Query class' `.aggregate_module` module attribute to allow Django ORM aggregate functions to work with MS SQL Server. The setting is enabled by default.

To disable the monkey patching, set the attribute to False. This will result in aggregate ORM functions provided by Django to fail with the MS SQL Server backend due to incompatibilities between the SQL code used by the Django implementation for these functions.

`mxodbc_monkey_patch_database_cache`

Enables monkey patching the Django `DatabaseCache` class to allow the Django ORM database cache functionality to work with MS SQL Server. The setting is enabled by default.

To disable the monkey patching, set the attribute to False. This will result in database cache ORM function provided by Django to fail with the MS SQL Server backend due to incompatibilities between the SQL code used by the Django implementation for the caching functionality.

`mxodbc_stringformat`

Set the mxODBC string format to use.

Default is to use mxODBC's `NATIVE_UNICODE_STRINGFORMAT`, since most drivers support Unicode today.

If a driver does not support Unicode, you have to set this to mxODBC's `UNICODE_STRINGFORMAT`.

The FreeTDS ODBC drivers prior to version 0.91 do not support Unicode. mxODBC Django Database Engine defaults to `UNICODE_STRINGFORMAT` for those drivers automatically. For other drivers, you may have to adjust the setting if you run into problems with Unicode data or SQL statements.

Please consult the *mxODBC User Manual* for additional details. Note that mxODBC itself defaults to `EIGHTBIT_STRINGFORMAT` for all drivers in order to stay backwards compatible with older mxODBC versions.

`mxodbc_timestampresolution`

Sets the mxODBC timestamp resolution to use on connections. The value is given in nanoseconds as integer.

Default is 1000000 nanoseconds, which corresponds to 1 millisecond. Using lower values can cause MS SQL Server to raise errors.

You can override that default by explicitly passing an integer nanosecond value here. Please consult the *mxODBC User Manual* for details.

Please note that you normally do not need to modify this value.

`mxodbc_use_executedirect`

Forces using of the mxODBC `cursor.executedirect()` method instead of the `cursor.execute()` method used normally.

This option is enabled by default for MS SQL Server database backends, since it results in better performance and allow working around some problems with the FreeTDS ODBC driver, but left disabled for all other database servers.

You can enable or disable it by setting the option to True or False.

Example:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {
            'mxodbc_encoding': 'cp1252',
            'mxodbc_exact_collation': 'Latin1_General_BIN',
            'mxodbc_inexact_collation': 'Latin1_General_CI_AS',
        },
    }
}
```

The syntax used for defining the OPTIONS dictionary is standard Python syntax.

5. Using the mxODBC Django Database Engine

There are two ways to connect to databases using the mxODBC Django Database Engine:

- *connecting using the Django ORM* and using the database under ORM control
- *connecting using the mxODBC API* and using the database using the DB-API 2.0 compatible interface

The following sections explain the details on both variants.

5.1 Using the Django ORM with mxODBC

The mxODBC Django Database Engine implements the ORM database engine API needed by the Django ORM to work with the database.

5.1.1 MS SQL Server as database backend for the Django ORM

The Django ORM engine implementing the ORM interface for MS SQL Server is called

```
mxodbc_django.ms_sql_server
```

This is the database engine name that you have to use for the `ENGINE` entry in the `DATABASES` setting of your Django projects settings.py module.

Example:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
    }
}
```

Once configured, you can use the Django ORM with the MS SQL Server backend data source as usual. Please see the *Django User Manual* for details on how to use the *Django ORM interface*, in particular the *Django model layer documentation*.

Database Permissions

If you intend to run `python manage.py syncdb` on the connection, please make sure that the database user you are connecting with has sufficient database permissions to create and alter tables.

Date/Time Fields

For maximum compatibility across MS SQL Server versions, the `mxodbc_django.ms_sql_server` engine uses SQL Server `datetime` fields to represent Django `DateField()`, `DateTimeField()` and `TimeField()`.[2]

The engine takes care of automatically converting between the Python datetime module values and the database values, i.e. a `TimeField()` will be returned as `datetime.time` object, even though the database stores it together with a 1970-01-01 date.

A side-effect of this is that the Min() and Max() aggregates on DateField() and TimeField() fields don't work as expected. See 6.1.4 Min() / Max() and DateField() / TimeField() for details.

MS SQL Server's `datetime` field supports a range of January 1, 1753, through December 31, 9999. Any other dates will cause an exception to be raised.

Seconds fractions are rounded to the nearest millisecond before passing the values to SQL Server. MS SQL Server itself only supports a resolution of 3.33 milliseconds and rounds these to the nearest increments of 0.000, 0.003, 0.007 seconds. See 6.1.3 Limited MS SQL Server datetime precision for more details.

Limitations

Please note that some minor limitations apply when using MS SQL Server as ORM backend. These are listed in section 6.1 Known problems and limitations of the MS SQL Server subpackage.

5.1.2 Working with databases which are not supported by the ORM

At the moment, we only support MS SQL Server as ORM backend in the mxODBC Django Database Engine.

[2] In future versions of mxODBC Django Database Engine, we plan to optionally make use of the `date` and `time` fields, that were added in SQL Server 2008.

If you want to access other database backends, you will either have to try using the MS SQL Server subpackage `mxodbc_django.ms_sql_server` of the mxODBC Django Database engine, or directly interface to the database using the mxODBC API, which provides a Python DB-API 2.0 with many extensions.

Please see the section 5.2 Direct mxODBC Database Interfacing for details, if you plan to use mxODBC directly from within Django.

5.2 Direct mxODBC Database Interfacing

The mxODBC Django Database Engine was written to aid in using mxODBC with the Django ORM. However, it is easily possible to also use the included mxODBC library directly for interfacing to database backends.

5.2.1 mxODBC Python API

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

If you want to learn more about the Python DB-API 2.0, which mxODBC implements, please have a look at the following resources:

- *Python PEP 249: The Python DB-API 2.0*
- *eGenix Talk: Introduction to Python Database Programming*

5.2.2 Importing mxODBC into your Django application

Importing mxODBC can be done from a Django application just like from within a normal Python script. The only difference is that you have to **import the `mxodbc_django` package before trying to import `mx.ODBC`.**

Importing the `mxodbc_django` package makes sure that mxODBC is configured correctly for use in Django. If you forget to import the package before importing `mx.ODBC`, you will get an `ImportError`.

5.2.3 Example of using the mxODBC Database Interface in Django

In this short example. we're highlighting some important details of using the mxODBC database interface in Django.

First is the way mxODBC is imported:

```
# Import the mxODBC Django Database Engine to setup mxODBC
# for use in Django
import mxodbc_django

# Now, import mxODBC as usual
import mx.ODBC.Manager as ODBC
```

It is important to note that the `mxodbc_django` package has to be imported prior to importing mxODBC itself.

Hint: Using the `mx.ODBC.Manager` subpackage allows your Django application to work in most configuration settings, since it removes the need to know which ODBC manager the system is using.

Next, you can open a data connection by providing a data source connection string to the `DriverConnect` constructor:

```
# Open a connection to a database
connection = ODBC.DriverConnect(connection_string)
```

Once you have the connection, you can create cursors on the connection to execute statements.

```
# Create a cursor to run SQL statements
cursor = dbc.cursor()

# Run a query
cursor.execute(query_sql, parameters)

# Fetch the results
results = cursor.fetchall()

# Insert/update some rows
cursor.executemany(insert_sql, list_of_parameters)
```

Whenever making changes to the database, you have to commit those changes explicitly to make them permanent in the database. If you don't, the changes will be rolled back again as soon as you close the connection.[3]

```
# Commit the changes that you have made to the database; If you
# forget this, your inserts won't be written permanently to
```

[3] If you setup an auto-commit connection, all changes will be permanent immediately, so you don't have to call `.commit()`. However, you also lose the possibility to easily undo changes.

```
# the database (unless you've setup an auto-commit connection)
connection.commit()
```

You often have auto-increment fields in databases, especially for primary keys. If you want to know which values have been created by the database, you need to fetch those values after the insert.

```
# Run another query, e.g. to check the inserts and fetch
# the generated primary keys
cursor.execute(query_sql, parameters)

# Fetch the results
results = cursor.fetchall()
```

Once you're done with using the database cursor, make sure you close the cursor to free up resources.

```
# Close the cursor after you've finished using it in order
# to free up resources
cursor.close()
```

The same has to be done with the connection. Note that closing the connection will do an implicit roll back of all uncommitted changes o the connection.

```
# Close the connection to the database to free up resources
connection.close()
```

5.2.4 Transaction Management

There is one important aspect to keep in mind when using the mxODBC Python API directly in your Django apps: The Django ORM uses its own, mostly automatic transaction management.

When using mxODBC directly, you will have to either integrate the connection transactions with the ORM or manage your own transaction management.

Default transaction mode is manual commit

In particular, **mxODBC does not default to auto-commit mode**, so you have to commit all changes to the database explicitly by calling `connection.commit()` (or `connection.rollback()` in case you want to revert the changes).

Without an explicit `connection.commit()` call, changes will not be written to the database. mxODBC defaults to rolling back the changes, if a connection is closed without a commit.

Enabling Auto-Commit

You can enable auto-commit on a connection by setting

```
connection.autocommit = True
```

after having connected to the database. However, this is not encouraged, since doing can easily create inconsistencies in your database if your Django application runs into an unexpected error.

To disable auto-commit, close all cursors on the connection and run:

```
connection.autocommit = False
```

6. Additional information

6.1 Known problems and limitations of the MS SQL Server subpackage

The Django internal database API has evolved quite a bit over time. Even though most of it is now written in a way that makes it portable between backends, there are some areas which still use hard-coded SQL or make assumptions that don't apply to all database backends.

In this section we list known problems and limitations with the Microsoft SQL Server subpackage **mxodbc_django.ms_sql_server** of the mxODBC Django Database Engine.

Most of the limitation arise from the fact that MS SQL Server is missing some features used in the Django ORM, or requires a different SQL dialect than the database backends natively supported by Django.

While we have tried to work around a couple of issues, some assumptions made in the ORM cannot easily be fixed without changing the ORM code and making it more portable.

6.1.1 mxODBC Django Database Engine currently only supports Microsoft SQL Server

mxODBC Django Database Engine currently only supports Django ORM integration for Microsoft SQL Server 2005, 2008 and 2012.

We will add support for other database servers in future releases.

Note that the Django ORM integration is available on all supported platforms, not only Windows, provided you have an ODBC driver available for the platform.

On Linux, we suggest looking at the official Microsoft SQL Server Native Client for Linux. On other Unix platforms, the FreeTDS ODBC driver or one of the many commercial drivers can be used.

6.1.2 Django timezone support doesn't work well with MS SQL Server

The ODBC API does not support passing or retrieving timezone aware date/time values. As a result the underlying mxODBC cannot easily support timezone aware datetime objects.

In order to maintain compatibility with Django 1.4 and later that support the USE_TZ setting, the mxODBC Django Database Engine ignores the `.tzinfo` attribute on datetime objects passed to the adapter. All date/time data read from the database will also not have the `.tzinfo` attribute set.

Avoid using USE_TZ

We recommend not using the **USE_TZ** setting when using MS SQL Server as database backend. It is best practice to store date/time values as Universal Time (UTC) in the database and to apply any locale dependent conversion in the UI layer of the application - based on user, browser, session or system preferences.

Avoid date/time string literals and implicit datetime to string conversion

Likewise try to avoid implicit conversion of datetime values to strings. MS SQL Server interprets such literal date/time values in a locale dependent way, which can lead to surprising results.

6.1.3 Limited MS SQL Server datetime precision

MS SQL Server datetime fields only have an accuracy of 3.33 milliseconds, or short 0.00333 second, not the microsecond accuracy of Python datetime objects. Furthermore, it rounds to the nearest increment of 0.000, 0.003, 0.007 seconds.

Unwanted Rounding

This can lead to unexpected behavior when querying date/time or time ranges, esp. when using non-inclusive upper limits in range queries or when inserting timestamps which could be subject to unwanted rounding.

Example:

Instead of `datetime(2008, 12, 31, 23, 59, 59, 999999)` you may have to use `datetime(2008, 12, 31, 23, 59, 59, 997000)` as date/time value, since using 999999 microseconds could cause rounding to `datetime(2009, 1, 1, 0, 0, 0, 0)`.

The mxODBC Django Database Engine uses datetime fields for Django **DateField(), DateTimeField(), TimeField()**, so the above applies to all those Django ORM fields.

Avoid mixed-date/time field type comparisons/filtering

For the same reasons as above, it is also good practice to not use mixed date/time comparisons or filtering, such as comparisons of a `DateField()` with a `DateTimeField()` in a filter.

Always try to use comparisons/filtering between same-type date/time fields to avoid introducing bugs into your application due to rounding, e.g. `DateFields()` with `DateField()`, or `DateTimeField()` with `DateTimeField()`.

6.1.4 Min() / Max() and DateField() / TimeField()

Because `DateField()` and `TimeField()` Django fields are mapped to SQL Server `datetime` fields, the SQL aggregate function MIN() and MAX() on these fields will return `datetime.datetime` objects when queried from the database.

As a result, the Django aggregate functions `Min()` and `Max()` will return `datetime.datetime` objects as well - even when used on `DateField()` or `TimeField()` fields.

Since the needed field information is not available to the ORM aggregate functions and Django doesn't provide a clean way to override them, this cannot be changed.

6.1.5 Character encoding related problems

When using the FreeTDS ODBC driver on Unix systems, applications may be limited to the character set of a specific encoding, since FreeTDS has only just started supporting Unicode natively in recent releases.

The following exceptions are examples of the exceptions you get whenever the applications tries to use characters that are not supported by the encoding:

```
OperationalError: ('HY000', 2402, "[unixODBC][FreeTDS][SQL
Server]Error converting characters into server's character set.
Some character(s) could not be converted", 7737)
```

```
UnicodeEncodeError: 'charmap' codec can't encode characters in
position 0-1: character maps to <undefined>
```

```
UnicodeEncodeError: 'charmap' codec can't encode character u'\x84'
in position 5: character maps to <undefined>
```

Some unit tests in the Django unit test suite fail due to this limitation when running on a Unix system with FreeTDS.

You can adjust the encoding used by the mxODBC Django Database Engine by setting the *mxodbc_connection_encoding* database option in your settings.py file:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {
            mxodbc_connection_encoding: 'cp1252',
        },
    }
}
```

Note that mxODBC Django Database Engine will automatically default to 'cp1252' when using FreeTDS on Unix and 'utf-8' when using the SQL Server Native Client ODBC driver on Windows.

6.1.6 Deferred constraint checking is not supported on MS SQL

There is no generic support for deferred checking of foreign key constraints in Microsoft SQL Server.

As consequence, mxODBC Django Database Engine does not allow using this feature and raises an exception whenever there are conflicts due to foreign key constraints, e.g. when trying to load data in the wrong dependency order.

The following exceptions are examples of the exceptions raised because of this:

```
IntegrityError: ('23000', 547, '[unixODBC][FreeTDS][SQL Server]The
DELETE statement conflicted with the REFERENCE constraint
"f_id_refs_id_52a6fc20". The conflict occurred in database
"test_django", table "dbo.delete_e", column \'f_id\'.', 7737)

IntegrityError: ('23000', 547, '[unixODBC][FreeTDS][SQL Server]The
INSERT statement conflicted with the FOREIGN KEY constraint
"FK__serialize__autho__3D5E1FD2". The conflict occurred in database
"test_django", table "dbo.serializers_author", column \'id\'.',
7737)
```

Some unit tests in the Django unit test suite fails due to this limitation.

Possible work around

If you must support forward references or cyclic references in your database schema, the only option is to define the respective columns as NULLable, insert the rows with NULLs (None in Python) in those columns and then set the values in a subsequent rounds of updates.

6.1.7 Limited support for regular expressions

Microsoft SQL Server 2005, 2008 and 2012 do not directly support the usage of regular expressions.

mxODBC Django Database Engine currently emulates the most trivial regular expression patterns using a stored function created when you create your database tables with the first *syncdb* operation.

Note that the user running your application will need execute privileges for that stored function if you use `__regex` or `__iregex` conditions in your code or one of the Django add-ons you use.

6.1.8 Data types nvarchar() and ntext cannot be compared

MS SQL Server does not support comparing nvarchar() and ntext columns for equality. The same is true for comparisons with other variable length column types (ntext, text, image).

With SQL Server 2012, Microsoft has decided to deprecate these variable size column types. See *this MSDN article* for details.

The Django ORM can trip over this limitation in the implementation of *generic relations*.

Note that the MS SQL Server subpackage of the mxODBC Django Database Engine does not use the above variable size column types, so you will likely only run into this situation when interfacing to database not under Django ORM schema control.

Work around

You can either avoid using text, ntext and image column types in the database schema or explicitly cast the variable size column types to limited size column types.

6.1.9 Aggregate function support conflicts with other database backends

In order to support the different SQL dialect used by MS SQL Server for aggregate functions, the mxODBC Django Database Engine has to patch the Django ORM Query class and provide a customized aggregates module so that requests for aggregates get converted to SQL code which is compatible with MS SQL Server.

Unfortunately, the Django ORM does not allow providing such customizations on a per-database backend basis.

The patching of the Django ORM can be disabled, if needed, but since we assume that most Django ORM users will use only one database backend per application, we have enabled the patching by default.

If you want to disable patching the Django Query class, you can do so by setting the *mxodbc_monkey_patch_aggregates* database option in your settings.py file to False:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {
            mxodbc_monkey_patch_aggregates: False,
        },
    }
}
```

Please note that by doing so, some aggregate functions used by the ORM will no longer work with the MS SQL Server backend and cause tracebacks.

6.1.10 Database cache support conflicts with other database backends

For the same reason as in the previous section concerning aggregate functions, the mxODBC Django Database Engine has to patch the Django ORM DatabaseCache class.

The Django ORM assumes that all database backends support the LIMIT/OFFSET SQL syntax for limiting the result set to a predefined window. MS SQL Server does not support this syntax, which is why we had to provide different queries for the cache implementation. We also had to make sure that the ORM only uses non-aware timestamps for the caching.

Unfortunately, the Django ORM does not allow providing such customizations on a per-database backend basis.

The patching of the Django ORM can be disabled, if needed, but since we assume that most Django ORM users will use only one database backend per application, we have enabled the patching by default.

If you want to disable patching the Django DatabaseCache class, e.g. because you want to use a different database backend as database cache, you can do so by setting the *mxodbc_monkey_patch_database_cache* database option in your settings.py file to False:

```
DATABASES = {
    'default': {
        'ENGINE': 'mxodbc_django.ms_sql_server',
        'NAME': 'exampledb',
        'DSN': 'DSN=sqlserver2008;UID=sa;PWD=123',
        'OPTIONS': {
            mxodbc_monkey_patch_database_cache: False,
        },
    }
}
```

Note that the MS SQL Server backend is not usable as database cache in Django when disabling the patch.

6.2 Tips and tricks

6.2.1 How do I determine the correct collation values?

Please consult to the manual of your database server for the list of valid collation names. You can find more information on the *Microsoft SQL Server MSDN Web site*.

You can list all the collations supported by Microsoft SQL Server using the *SQL Server Management Studio* GUI administration tool. The complete list is shown when starting the *Create Database* wizard under the *Options* tab.

The SQL Server Management Studio will also show you the currently set database default collation in the database properties.

6.2.2 How do I implement full regular expression search?

MS SQL Server does not have native support for regular expression, so the only option you have is to work around this by filtering the records using the Python `re` *module*.

6.3 Troubleshooting

6.3.1 Django does not find the database backend

Please make sure you installed mxODBC Django Database Engine into the same Python instance as your Django based application uses.

Try to open the command line interpreter of the same Python instance and import the subpackage from mxODBC Django Database Engine manually with a single import statement and watch for an ImportError.

Make sure you did not mistype the name of the mxODBC Django Database Engine subpackage and that you entered its full (dotted) module name, not just the name of the subpackage. See the example settings above.

6.3.2 I'm getting an error about missing mxODBC license or the license has expired

Please make sure you have a valid mxODBC license installed.

Please visit the *eGenix.com* web-site to obtain a license.

6.3.3 Django cannot connect to the database

Please make sure

- you configured the ODBC data source correctly (please check the *mxODBC User Manual* for details)
- you specified the name of the database, host and port in your ODBC data source definition (please see your ODBC driver's documentation for details)
- you used the correct data source name in the DSN engine setting (`DSN=<data source name>`)
- you defined the user name and password correctly in the DSN engine setting (`UID=<username>;PWD=<password>`)
- your database server is up and running and accepts incoming connections (the type of the connection and the address should

match the values you entered into the ODBC Manager when you defined your DSN)

- all firewalls between the Django server and the database server permit traffic on the required ports in both directions

In order to debug the problem, please

- review the Django logs for hints
- enable ODBC manager logging and check the ODBC manager logs for hints
- review the logs of the database server for details, enable logging if required, please consult your database server's documentation for details

7. 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*.

8. Copyright & License

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

> Please note that using this software in a commercial environment 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.com mx Extensions Page* 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*.

EGENIX.COM COMMERCIAL LICENSE AGREEMENT

Version 1.2.0

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 as follows:

1) "CPU" refers to a central processing unit which is able to store and/or execute the Software (a server, personal computer, 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.

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. Likewise, 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>