NetLogo SQL Wrapper User manual

Jan Blom
Rene Quakkelaar
Mark Rotteveel

NetLogo SQL Wrapper User manual

by Jan Blom, Rene Quakkelaar, and Mark Rotteveel

Covers NetLogo SQL Wrapper version 1.1

Preface

1. Introduction to NetLogo SQL Wrapper

NetLogo SQL Wrapper is an extension to NetLogo which adds primitives to the NetLogo modelling language to support access to databases using SQL (Structured Query Language).

The following features are supported:

- Querying data from a database (**SELECT**)
- Storing data into a database (**INSERT**)
- Changing data in a database (**UPDATE**, **DELETE**)
- Database connection per agent (turtle, patch)
- (optional) Automatic connection management (creates connections when needed, and closes when possible)
- (optional) Transaction control

SQL Wrapper is released under the LGPL. The source code is available at http://code.google.com/p/netlogo-sql/.

The extension was commissioned by Dr. Hendrik Drachsler, Assistant Professor at the Centre for Learning Sciences and Technologies (CELSTEC) of the Open University of the Netherlands (OUNL). Design and development was done by Jan Blom, Rene Quakkelaar and Mark Rotteveel as part of the final project of their Bachelor (BSc Computer Science) at the Open University of the Netherlands (OUNL).

2. Extension Requirements

SQL Wrapper requires NetLogo 4.1 or 4.1.x in combination with Java 6.

Currently SQL Wrapper supports the following databases:

MySQL	4.1, 5.0, 5.1, 5.4, 5.5
PostgreSQL	8.x, 9.x

SQL Wrapper also provides a generic database option to use JDBC drivers to connect to database systems not explicitly supported.

3. This manual

This manual describes the language primitives the extension adds to NetLogo, how to use them and more advanced information on usage and pitfalls. We assume that readers of this manual and users of SQL Wrapper are already knowledgeable on NetLogo and SQL, including knowledge on designing databases (tables etc).

Part I. Getting Started

Chapter 1. Installation

Before SQL Wrapper can be used, it needs to be installed in the `extensions` folder of NetLogo. We assume that NetLogo itself is already installed, and that you know where its program folder is located. On most systems you will also need sufficient rights to that program folder (eg Administrator rights on Windows) to be able to install SQL Wrapper.

The SQL Wrapper is distributed as a single ZIP-file, `sql-bin-<version>.zip`, and contains all files necessary for normal extension usage. Installation is as simple as the following three steps:

1. If NetLogo is running, stop it first
2. If a previous version of SQL Wrapper (folder `sql`) is installed: delete it from the `extensions` folder.
3. Unzip `sql-bin.zip` to the NetLogo `extensions` folder (and make sure that the folder structure is preserved)

   On Windows: Use a program like WinZIP to unzip the file to `C:\Program Files\NetLogo 4.1\extensions`

   On Linux : Use the program "unzip" to unzip the file to the `extensions` subdirectory of the NetLogo installation directory
4. (optional) Add the JDBC drivers for your database into the `extensions/sql` subdirectory, see *Connecting to other databases*

   ### Note

   This step is only required for use with the generic database option, or for database drivers that are not included in the SQL Wrapper distribution for legal reasons
5. Start NetLogo

Chapter 2. Tutorials

1. First Query

To check if the extension is installed and working correctly, you can use the following example of use. To keep this example as simple as possible, we will use an empty NetLogo model and use the Command Center to execute the commands.

To follow this example you will need access to a MySQL database, and have `SELECT` rights to a table containing one or more rows of data. For the rest of this section we will assume your MySQL connection details are: server `localhost`, port `3306`, catalog (database) `default`, user `root`, password `testpassword`, table `testtable`.

This example runs in an empty model, so first we will create a new model through File, New (save or discard your current model as you see fit).

Next we will need to instruct NetLogo to use the SQL Wrapper extension, which has the extension-name `sql` in NetLogo. To do this, go to the Procedures tab and instruct NetLogo to import the extension:

```
extensions [sql]
```

Now switch to the Interface tab. If everything is OK and the extension was found and loaded, NetLogo will now display the Interface tab. If the extension can't be found, the Procedures tab will remain open, displaying an error-message `Can't find extension: sql`, if this occurs, please check *Installation*. For syntactic errors, it will display a message like `Expected <syntactic element>`.

Now that the extension has been loaded, we can use the Command Center to run commands. First we will need to instruct the extension how to connect to the database. To give the connection details to SQLWrapper, enter on the Observer prompt:

```
sql:configure "defaultconnection" [["host" "localhost"] ["port" 3306] ["user" "root"]
 ["password" "testpassword"] ["database" "default"]]
```

The command above instructs SQL Wrapper to connect to schema `default` on `localhost`, port `3306` using user name `root` and password `testpassword`. If the wrong database information was used, NetLogo will show a popup with the title Runtime Error containing the details of the error. The **sql:configure** command with the Aspect: "defaultconnection" enables automatic connection management, which will allocate connections when needed.

For this tutorial we also assume a table `testtable` exists with columns `ID` (type: INTEGER; primarykey), `NAME` (type: VARCHAR(50)) and `INFO_DATE` (type: DATETIME).

Now that SQL Wrapper knows how to connect to the database, we can execute a simple query to create the actual connection and execute the query:

```
sql:exec-query "SELECT * FROM testtable WHERE ID = ?" [1]
```

The command **sql:exec-query** can be used to execute SELECT queries with parameters. The first argument is a string and contains the query to execute, in this query a question mark (`?`) is used to indicate a parameter to the query. The second argument is a list. This list contains the values that need to be substituted for the parameters. These values are converted to the right datatype in the database, and are - if necessary - escaped.

To retrieve the result, there are two options **sql:fetch-row** or **sql:fetch-resultset**. The first returns a single NetLogo list containing the values of a single row, it is suitable for row by row processing. The second returns all rows (as a NetLogo list containing a list with the values of a row. For changes to the database (**INSERT**, **UPDATE**, **DELETE**) the command **sql:exec-update** can be used.

To fetch a single row:

```
show sql:fetch-row
```

This returns a single row, for example looking like this:

```
observer: [1 "Example" "2010-10-31 11:39:10.0"]
```

Now that we have connected to the database and run our first query, we have reached the end of this tutorial.

Part II. Reference

The reference section contains information on the syntax and usage of the commands and reporters that SQL Wrapper makes available to NetLogo.

Chapter 3. Importing the Extension

NetLogo imports extensions using the `extensions` keyword, followed by a list of extension-names. The extension-name of SQL Wrapper is `sql`, importing the extension into a model can be done with the following instruction:

```
extensions [sql]
```

If the extension can't be found, NetLogo will display the error-message `Can't find extension: sql`.

More information on extensions can be found in the NetLogo manual, Extensions Guide.

Chapter 4. Configuring the extension

1. sql:configure

The **sql:configure** command is used to configure aspects of the plug in.

```
sql:configure aspect [parameters...]
```

aspect
: Name of the aspect to configure (see sub-sections below)

parameters...
: List containing the configuration for the aspects (see sub-sections below). Every parameter is a list with key-value-pairs (a list per pair). In general this will look like `[["parametername" parametervalue] ... ]`.

Configuration should normally only be done in the setup-phase of a model.

1.1. Aspect: "defaultconnection"

The aspect *`"defaultconnection"`* configures the database connection information for the connection pool and (re)initializes the connection pool. The configuration options for this aspect are discussed in more detail in Connection pool (or automatic connection management)

1.2. Aspect: "connectionpool"

The aspect *`"connectionpool"`* configures the connection pool itself (eg timeout, number of connections).

```
sql:configure "connectionpool" [["timeout" timeout] ["max-connections" max-connections]
 ["partitions" partitions]]
```

timeout
: sets the timeout for obtaining a connection in seconds (type: integer). Defaults to `5` seconds if the "timeout" list is not passed. Setting to `0` disables timeout (this can freeze the model if no connections are released).

max-connections
: maximum number of database connections that the connection pool should make available (type: integer). Defaults to `20` if the "max-connections" list is not passed.

  Be aware that the database server could have additional limits imposed (eg maximum number of connections per user, or per IP): these limits are not taken into account. Please check with your database server administrator if these limits apply.

partitions
: number of partitions in the connection pool (type: integer). Optional, defaults to `1` if the "partitions" list is not passed.

  This setting is for advanced configuration of the connection pool. It is normally not needed (nor advisable) to set this to anything other than 1.

  It is advisable to make the value of max-connections above divisible by the number of partitions (integer division will lead to a lower number of maximum connections). The number of connections per partition should be 2 or higher.

The defaults specified above will be applied if the *`"connectionpool"`* aspects is not configured explicitly. Be aware that these settings will only be applied at the next configuration of Aspect: "defaultconnection" and the settings are not validated until that time.

1.3. Aspect: "logging"

The aspect *`"logging"`* configures logging that is done by the plug in, and through the **sql:log** command. SQL Wrapper handles its own logging, through the Java logging facility (java.util.logging). By default, all logging is

switched off. If logging is needed, it might be helpful to consult the Java documentation for logging in addition to the information given here.

```
sql:configure "logging" [["path" path] ["file-logging" file-logging] ["level" log-level]
 ["copy-to-stderr" copy-to-stderr]]
```

path
: path of the folder were logfiles will be stored. Defaults to `null` (not configured) if "path" list is not passed in. The user must of write access and the folder must exist.

  The path can start with a prefix: `%c` indicates the current folder, `%m` the folder that stores the current NetLogo model.

  Also, the substitutions that apply for java.util.logging.FileHandler apply, like `%t` for the system temporary directory. See the Java API documentation on logging for more information. (Available at http://download.oracle.com/javase/6/docs/api/java/util/logging/FileHandler.html) When path is set, file-logging is turned on implicitly (see below)

file-logging
: enable or disable the logging to file (type:string): `on` or `true` (enable), `off` or `false` (disable). Default `off`.

  The file-logging should be configured by setting the path option (see above).

log-level
: The minimum level of the messages that will actually be written. The available levels are, from high to low: `"SEVERE"`, `"WARNING"`, `"INFO"`, `"FINE"`, `"FINER"` and `"FINEST"`, or everything: `"ALL"`. The levels are interpreted case-insensitive, so `"SEVERE"`, `"severe"` and `"Severe"` are all valid. Default is `"ALL"`.

  When an invalid level is specified, the currently set level remains set, no error is given.

copy-to-stderr
: enable or disable copying of logging to the standard error output (type: string): `on` or `true` (enable), `off` or `false` (disable). Default `off`.

  When switched on, all logging is copied and written to the system error stream (stderr on Unix/Linux). Other logging settings are not affected by enabling or disabling this options.

The specified defaults will be applied when no explicit configuration of logging is performed.

2. sql:get-configuration

The **sql:get-configuration** retrieves the current configured values of an aspect of the plug in.

```
sql:get-configuration aspect
```

aspect
: Name of the aspect

The return value is a list with the first entry being the name of the aspect and the rest of the entries are the key-value-pairs of the configuration (a list per pair).

3. sql:get-full-configuration

The **sql:get-full-configuration** retrieves the current configured values of all aspects of the plug in.

```
sql:get-full-configuration
```

The return value is a list with a list per aspect in the format described in **sql:get-configuration**.

Chapter 5. Connecting to a Database

1. Connection pool (or automatic connection management)

It is advisable to use the connection pool provided by Aspect: "defaultconnection". The connection pool simplifies working with connections, especially when combined with the *autodisconnect* option.

The aspect *"defaultconnection"* configures the database connection information for the connection pool and (re)initializes the connection pool.

```
sql:configure "defaultconnection" [["brand" brandname] ["host" hostname] ["port" port]
 ["database" catalog] ["jdbc-url" jdbc-url] ["driver" driver-class] ["user" username]
 ["password" password] ["autodisconnect" autodisconnect]]
```

brandname
: name of the database type (type: string). Optional, defaults to `MySQL` if the "brand" list is not passed.

  Valid values: `MySQL`, `generic`, `PostgreSQL` (case insensitive)

hostname
: hostname of the database server (type: string). Optional, defaults to `localhost` if the "host" list is not passed.

  Applies to: `MySQL`, `PostgreSQL`

port
: port number of the database server (type: integer). Optional, defaults to `3306` for `MySQL` and `5432` for `PostgreSQL` if the "port" list is not passed.

  Applies to: `MySQL`, `PostgreSQL`

catalog
: name of the database catalog (type: string)

  Applies to: `MySQL`, `PostgreSQL`

jdbc-url
: URL to the database, using the JDBC (Java DataBase Connectivity) format for your database (type: string). Required for `generic`.

  Applies to: `generic`

driver-class
: Name of the JDBC driver class to load (type: string). Required for `generic`.

  For `MySQL` defaults to `com.mysql.jdbc.Driver`

  For `PostgreSQL` defaults to `org.postgresql.Driver`

username
: Username to the database (type: string)

password
: Password to the database (type: string)

autodisconnect
: configure automatic disconnect (type: string). Optional, defaults to `on` if the "autodisconnect" list is not passed. Valid values: `off` or `false` (disable autodisconnect), `on` or `true` (enable autodisconnect)

  The automatic disconnect will close connections as soon as possible (this returns them to the connection pool for use by other agents). With this option enabled, a connection will be closed after executing an update

(**UPDATE**, **INSERT**, **DELETE**) or after fetching the full resultset (**fetch-resultset**) or the last row of a resultset (**fetch-row**) of a **SELECT** when in autocommit mode, or after performing a commit or rollback when using transactions.

Use of autodisconnect is strongly recommended for models using a large number of agents, because otherwise the number of required connections will be high (and could exceed the maximum number of connections supported by the database). Autodisconnect is only applied to connections obtained from the connection pool. Connections created using **sql:connect** will not be disconnected automatically.

When the automatic connection management (or connection pooling) has been configured, all commands and reporters will obtain a connection from the connection pool if no explicit connection has been created using **sql:connect**. A connection is associated with the current agent until the connection is closed using **sql:disconnect** or automatically disconnected when autodisconnect is enabled.

For more detailed information on the generic brand configuration, refer to *Connecting to other databases*

2. sql:connect

Create a connection to a database for the current agent.

```
sql:connect [["brand" brandname] ["host" hostname] ["port" port] ["database" catalog]
 ["jdbc-url" jdbc-url] ["driver" driver-class] ["user" username] ["password" password]]
```

The parameters of **sql:connect** are a subset of the parameters of Aspect: "defaultconnection". For a full description see Connection pool (or automatic connection management).

The **sql:connect** command can be used to create a connection for a specific agent (the agent in the current context). When connection pooling is used, this command is not necessary. Connections created using **sql:connect** are not subject to *autodisconnect* and need to be closed manually using **sql:disconnect**.

3. sql:disconnect

Disconnect the current agent from the database.

```
sql:disconnect
```

For connection pooling, the connection is returned to the connection-pool for re-use. When using connection pooling, it is advisable to use *autodisconnect* (default), or make sure connections are closed as soon as possible, so it can be re-used by another agent.

Important

Close a connection as soon as possible to limit the number of connections required.

4. sql:is-connected?

Checks if a connection is established.

```
sql:is-connected?
```

This reporter returns true if a connection is established, false otherwise. When connection pooling has been configured it will always return true. The command **sql:debug-is-connected?** can be used to check the actual connection status for pooled connections.

This reporter does not autodisconnect.

5. sql:use-database

Switches the connection to another database schema on the same server.

```
sql:use-database schema
```

schema
: name of the database schema (type: string)

This command can only be used on connections created using **sql:connect**; it will result in an error when executed on a connection from the connection pool.

Note

sql:use-database currently only works on `MySQL` connections, not for `generic` or `PostgreSQL`.

6. sql:current-database

Returns the name of the current database catalog name.

```
sql:current-database
```

This reporter does not autodisconnect.

Note

If sql-wrapper is unable to get the database catalog name, `default` is returned.

7. sql:find-database

Checks if the specified database catalog exists on the database server.

```
sql:find-database catalog
```

catalog
: name of the database catalog (type:string)

This reporter does not autodisconnect.

Note

sql:find-database currently only works on `MySQL` connections, not for `generic` or `PostgreSQL`.

8. sql:show-version

Prints the version of the SQL Wrapper plug in.

```
sql:show-version
```

This reporter will return a list with version information of the plug in.

Chapter 6. Executing SQL

1. sql:exec-query

The **sql:exec-query** command enables execution of parameterized SQL data-retrieval queries (like **SELECT**) which produce a resultset.

```
sql:exec-query parameterized-sql-statement [parameters...]
```

parameterized-sql-statement
: SQL statement that uses the `?` as a placed holder for parameters.

parameters...
: List of parameters that should be substituted for the `?` in the query. Every question mark in the parameterized query should have a corresponding parameter in the list.

A parameterized statement is a SQL statement with place holders - the question mark (?) - to indicate parameters. These parameters will be substituted with the values from the list of values in the second argument. SQL Wrapper accepts parameters of type string, number and boolean. Other NetLogo datatypes are not supported and will result in an error. The substitution is positional, which means that the first question mark in the query is replaced with the first value in the list. This also means that the number of parameters (question marks) in the query, and the number of values in the parameter-list must be the same.

The supported datatypes will be automatically converted to the equivalent type in the database and are escaped if necessary. Be aware that NetLogo uses a floating point datatype (double) for all numbers, so loss of precision can occur for fractional values and (large) integer values.

Table 6.1. Conversions from NetLogo to SQL

TO \ FROM	number	string	boolean
CHAR, VARCHAR, LONGVARCHAR	yes	yes	database dependent
NUMERIC, DECIMAL	yes	not supported	not supported
TINYINT, SMALLINT, INTEGER, BIGINT	yes	not supported	not supported
REAL, FLOAT, DOUBLE	yes	not supported	not supported
BOOLEAN	not supported	not supported	yes
BIT	not supported	not supported	not supported
TIME	not supported	not supported	not supported
DATE	not supported	not supported	not supported
TIMESTAMP	not supported	not supported	not supported
BINARY, VARBINARY, LONGVARBINARY	not supported	not supported	not supported
CLOB	yes	yes	not supported
BLOB	not supported	not supported	not supported

For conversions marked as *database dependent*, it is best to consult the manual of your database. An example: for he conversion of a NetLogo boolean value into a SQL CHARACTER type, MySQL uses `0` and `1`, and PostgreSQL uses `false` and `true`. Conversions marked as *not supported* are formally not supported by the plug in, but they might work depending on the value and/or database specific conversion rules.

For example MySQL will allow the number `105400` for TIME and convert it to `10:54:00`, but `106300` is not allowed as the minutes should be lower than 60, on the other hand PostgreSQL refuses this conversion. Similar

rules are available for DATE and TIMESTAMP, and for the conversion from string. In general it is advisable to use explicit SQL casts for these datatypes.

Before statements can be executed, the connection-pool needs to be setup or a connection needs to be created using **sql:connect**. The command **sql:exec-query** always creates a resultset, *Retrieving results* discusses processing of resultsets.

This command performs autodisconnect if the query has an empty result for pooled connections with autodisconnect enabled. Otherwise, autodisconnect will be applied when the resultset has been fetched completely. Autodisconnect will not be applied if autocommit is off.

2. sql:exec-update

The **sql:exec-update** command enables execution of parameterized SQL DML statements (like **UPDATE**, **INSERT** and **DELETE**) which do not produce a resultset.

```
sql:exec-update parameterized-sql-statements [parameters...]
```

parameterized-sql-statement
: SQL statement that uses the `?` as a placed holder for parameters.

parameters...
: List of parameters that should be substituted for the `?` in the query. Every question mark in the parameterized query should have a corresponding parameter in the list.

For more information on parameterized statements, see **sql:exec-query**.

Before statements can be executed, the connection-pool needs to be setup or a connection needs to be created using **sql:connect**. The command **sql:exec-update** never creates a resultset, the reporter **sql:get-rowcount** can be used to see how many rows were modified.

This command performs autodisconnect for pooled connections with autodisconnect enabled. Autodisconnect will not be applied if autocommit is off.

3. sql:exec-direct

The **sql:exec-direct** command allows execution of all types of SQL. Usually, the **sql:exec-query** and **sql:exec-update** commands are preferred.

```
sql:exec-direct sql-statement
```

sql-statement
: SQL statement to execute (type: string)

Before any queries can be executed, the connection-pool needs to be setup or a connection needs to be created using **sql:connect**. The reporter **sql:resultset-available?** can be used to check if the query produced a resultset. *Retrieving results* discusses processing of resultsets. If no resultset is available, the reporter **sql:get-rowcount** can be used to see how many rows were modified.

This command performs autodisconnect for pooled connections with autodisconnect enabled. If the executed query did produce a resultset, autodisconnect will only be applied if the resultset is empty, otherwise, autodisconnect will be applied when the resultset has been fetched completely. For other statements autodisconnect is applied immediately. Autodisconnect will not be applied if autocommit is off.

4. sql:resultset-available?

Checks if a resultset is available for processing.

```
sql:resultset-available?
```

This reporter can be used to check if the SQL statements produced a resultset (for example after **SELECT**). Returns `true` if the previously executed statement produced a resultset, `false` otherwise.

Use of this reporter is only necessary if you do not know in advance if the executed statement is a query or update statement.

5. sql:get-rowcount

Returns the number of rows modified (or added) by the last query.

```
sql:get-rowcount
```

Some SQL statements modify rows (for example **INSERT**, **DELETE** and **UPDATE**). This reporter will return the number of affected rows. If no rows were affected, 0 is returned. If there is no connection, no statement was executed, or if the executed statement could not modify the database (eg **SELECT** statements), -1 is returned.

Chapter 7. Retrieving results

1. sql:fetch-row

Fetches a single row of the result of a query.

```
sql:fetch-row
```

Calling **sql:fetch-row** again will return the next row, when there are no more rows available an empty list will be returned. If no connection is open, or if **sql:fetch-row** is executed when no resultset is available, an empty list will be returned as well.

Note

A single resultset should be processed either with **sql:fetch-row**, or **sql:fetch-resultset**, not a combination of both. Combining calls to these reporters for a single resultset will return empty lists, even if there are more rows to process.

For example: calling **sql:fetch-row** and then **sql:fetch-resulset** will return an empty list even though there might be more rows available.

This command will autodisconnect if autodisconnect is enabled, the connection was obtained from the connection pool, and the fetched row is the last row of the resultset.

The row is returned as a NetLogo list with the values of the row. The SQL datatypes are converted to their NetLogo equivalent:

Table 7.1. Conversions from SQL to NetLogo

FROM (SQL)	TO (NetLogo)
CHAR, VARCHAR, LONGVARCHAR	string
NUMERIC, DECIMAL	number
TINYINT, SMALLINT, INTEGER, BIGINT	number
REAL, FLOAT, DOUBLE	number
BIT, BOOLEAN	boolean
TIME	string (hh:mm:ss)
DATE	string (yyyy-mm-dd)
TIMESTAMP	string (yyyy-mm-dd hh:mm:ss.fffffffff where fffffffff indicates nanoseconds)
BINARY, VARBINARY, LONGVARBINARY	not supported (will return a string)
CLOB	string
BLOB	not supported (will return a string)

Some SQL datatypes are not formally supported, but the extension will attempt to return a string. As the NetLogo number datatype is based on a 64 bit floating point type, loss of precision can occur for decimal values or very large integer values. When conversion fails, an error will be thrown. This can for example occur if the database is able to store values that are not valid in Java (eg a MySQL DATE with value 0000-00-00 cannot be converted to a valid date in Java).

2. sql:fetch-resultset

Fetches all rows of the result of a query.

```
sql:fetch-resultset
```

This reporter will fetch all rows from the result of a query. The rows are returned as a NetLogo list with a NetLogo list per row, as described in **sql:fetch-row**.

Calling **sql:fetch-resultset** after calling **sql:fetch-row** (see note), or calling for a second time will return an empty list. Calling this command when the connection is closed, or when no resultset is available will return an empty list as well.

This command will autodisconnect if autodisconnect is enabled, and the connection was obtained from the connection pool.

3. sql:row-available?

Indicates if a next row is available in the resultset.

```
sql:row-available?
```

This reporter will return `true` if the next call to **sql:fetch-row** will return a row, or **sql:fetch-resultset** will return at least one row. This reporter will return `false` if no connection is open, there is no resultset, or the last row of the resultset has been read. This reporter can be used as a loop-condition when reading a resultset, see Looping over a resultset for an example.

Chapter 8. Transaction control

By default individual queries or updates are treated independent of each other (the so called auto-commit mode). For some applications, it is important that a group of queries and/or updates are executed atomically (that is: they should succeed or fail as one). This is were transactions come in: transactions allow a group of queries and updates to be committed or rolled back as one.

Important

For MySQL it is important to realise that transaction control only works for tables maintained by the InnoDB engine. Tables in MyISAM format are not subject to transaction control. If you want to use transactions in MySQL, please make sure that all your tables are in InnoDB format.

1. sql:autocommit-on

Enables auto-commit mode on the current connection.

```
sql:autocommit-on
```

Enabling autocommit on a connection will commit any active transaction.

2. sql:autocommit-off

Disables auto-commit mode on the current connection.

```
sql:autocommit-off
```

This command essentially does the same as **sql:start-transaction**.

3. sql:autocommit-enabled?

Reports on the autocommit status of the current connection.

```
sql:autocommit-enabled?
```

Returns `true` if autocommit is enabled, `false` if autocommit is disabled.

4. sql:start-transaction

Disables auto-commit mode on the current connection.

```
sql:start-transaction
```

Indicates start of transaction, the actual transaction is started when the first query or update is executed on this connection. During a transaction on a pooled connection with autodisconnect, the autodisconnect will only be performed for **sql:commit-transaction** and **sql:rollback-transaction**.

5. sql:commit-transaction

Commits the current transaction.

```
sql:commit-transaction
```

The commands commits the current active transaction. This will return an error if autocommit mode is on. After transaction commit, on connections without autodisconnect, a new transaction is started implicitly (so autocommit stays off). On connections with autodisconnect, the connection will be closed.

6. sql:rollback-transaction

Performs a rollback of the current transaction.

```
sql:rollback-transaction
```

The command performs a rollback of the current active transaction, undoing changes performed in the transaction. This will return an error if autocommit mode is on. After transaction rollback, on connections without autodisconnect, a new transaction is started implicitly (so autocommit stays off). On connections with autodisconnect, the connection will be closed.

Chapter 9. Logging and debugging

This chapter contains commands and reporters that can be used for logging or debugging purposes. Logging itself can be configured using the configure aspect described in Aspect: "logging".

1. sql:debug-is-connected?

Checks if a connection is established, without obtaining a connection from the connection pool.

```
sql:debug-is-connected?
```

The normal **sql:is-connected?** reporter will always report `true` if the connection pool is available. This reporter reports the actual connection status. This reporter is intended for use by the SQL Wrapper developers for testing and troubleshooting.

This reporter does not autodisconnect.

2. sql:log

Log information to the SQL Wrapper logfile.

```
sql:log level message
```

level
: log level to use (type: string). Valid values (case-insensitive): `SEVERE` (highest), `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, `FINEST` (lowest). Invalid values will be logged as `INFO`.

message
: message to write to the logfile

Part III. Advanced topics

The advanced topics part contains more detailed examples of usage inside NetLogo, and information on subjects that could impact developing a model that makes intensive use of database connections.

Chapter 10. Using SQL Wrapper in a NetLogo model

1. Looping over a resultset

The following shows a simple example for looping over a resultset and printing the rows.

```
; loops over TABLE testtable
to loop_t1
  sql:connect [["host" "localhost"] ["port" 3306] ["user" "root"] ["password"
 "testpassword"] ["database" "default"]]
  sql:exec-direct "SELECT * FROM testtable"
  while [sql:row-available?] [
    let row sql:fetch-row
    print row
  ]
  sql:disconnect
end
```

This example uses the **sql:row-available?** reporter to loop over the resultset and then fetch individual rows with **sql:fetch-row**. Finally the connection is closed using **sql:disconnect**.

2. Connection per agent

The example below demonstrates a case were the observer creates agents based on information in the database, and agents are asked to save themselves into the database.

When the agents are loaded only one connections is created: for the observer, when the agents are saved every agent will get its own connection to save itself.

```
extensions [sql]

breed [example_agents example_agent]

example_agents-own [
  db_id         ; Value of id in database
  sample_value  ; Value of sample_value in database
  in_db         ; true: saved in database (otherwise 0)
]

; Setup the example_agents by loading them from the database
to setup
  clear-all
  ; Define connection pool (autodisconnect disabled)
  sql:configure "defaultconnection" [["user" "root"] ["password" "testpassword"]
 ["database" "default"] ["autodisconnect" "off"]]

  ; Load agent data from database
  sql:exec-query "SELECT id, sample_value FROM example_agent" []
  while [sql:row-available?] [
    let agent-data sql:fetch-row
    create-example_agents 1 [
      set db_id (item 0 agent-data)
      set sample_value (item 1 agent-data)
      set in_db true
    ]
  ]
  ; autodisconnect is disabled, so manually disconnect as soon as possible
  sql:disconnect
end

; Save agents to the database
to save
```

```
  ask example_agents [
    ifelse in_db = 0 [
      ; First save: INSERT
      sql:exec-update "INSERT INTO example_agent (sample_value) VALUES (?)" (list
 sample_value)
      ; Retrieve value of id in database
      sql:exec-query "SELECT LAST_INSERT_ID()" []
      ; Update id with value from db
      set db_id (item 0 sql:fetch-row)
      ; Agent now stored in database
      set in_db true
    ][
      ; Already exists: UPDATE
      sql:exec-update "UPDATE example_agent SET sample_value = ? WHERE id = ?" (list
 sample_value db_id)
    ]
    ; autodisconnect is disabled, so manually disconnect as soon as possible
    sql:disconnect
  ]
end

; Create a new agent with random sample value
to create_agent
  create-example_agents 1 [
    set sample_value (random 200)
  ]
end

; Change sample_value of all example_agents with a random (positive or negative) delta
to random_change
  ask example_agents [
    let delta ((random 101) - 50)
    set sample_value sample_value + delta
  ]
end
```

The **setup** command first configures the connectionpool (and for this example we disable the autodisconnect option; see comments on the **save** command). The observer then queries a table called `example_agent` (with columns `id` and `sample_value`). For every row in the table, it creates a turtle of the breed example_agents and assigns the values of id and sample_value to agent-variables. Finally the connection is closed because we need to do this manually when autodisconnect is turned off.

The **save** command asks every agent to save itself. If the example_agent has not been saved yet it will **INSERT** itself into example_agent and then retrieve the new value for its id (db_id) and mark itself as saved. If the agent was already saved, it will use an **UPDATE** statement to update its sample_value in the database. Finally we close the connection manually as autodisconnect is disabled.

In the **setup** command we disabled autodisconnect, as the new id value needs to be retrieved after an **INSERT** (in this example) and we need to be sure we retrieve the right value. Normally when autodisconnect is enabled, the connection would have been returned to the connection pool after **sql:exec-update**. The following **sql:exec-query** would have retrieved a new connection from the connectionpool, and this may be a different connection. We could also have kept autodisconnect enabled, and used transactions instead as transactions disable autodisconnect for the duration of the transaction.

The other two commands, **create_agent** and **random_change** are intended to try out this model from the observer prompt of the model. The **create_agent** command creates a new - unsaved - example_agent with a random sample_value. The **random_change** command will update the sample_value of all example_agents with a random value between -50 and 50.

Chapter 11. Connections per agent, minimizing resource use

A NetLogo model can contain a large number of agents, while at the same time a database server may allow only a limited number of connections. This chapter discusses some strategies to reduce the number of connections needed. First we discuss how agents in NetLogo execute commands, and the impact that has on the number of connections required. Second we discuss the connectionpooling and autodisconnect. Then we show how transaction impact the number of connections. Finally we discuss methods of minimizing connections required, and configuring the extension to allow for more connections.

1. Ask and ask-concurrent

One of the important features of NetLogo, is the option to have agents execute commands for themselves. It is important to realise that NetLogo has only one thread of execution, this means that instructions (commands, reporters) are executed one at a time. The NetLogo has two commands to instruct agents to execute commands: **ask** and **ask-concurrent**. The command **ask** will instruct every agent in turn to execute all commands in the command block. This also means that the next agent will only start executing if the previous agent has completed all commands in the block. On the other hand, the **ask-conncurrent** command produces simulated concurrency by interleaving the commands: as soon as an agent completed a command that changed state (moved the turtle, changed a global, turtle, link or patch variable), another agent gets a turn to execute commands and so on. For more details on **ask** and **ask-concurrent**, see the NetLogo documentation.

For the next discussion we use an example without connection pooling or autodisconnect:

```
ask/ask-concurrent <agents> [
  sql:connect <connection-list>
  ...
  set <agent-variable> <new value>
  ...
  sql:disconnect
]
```

This example is fairly straightforward for **ask**: the first agent performs `sql:connect - ... - set <agent-variable> <new value> - ... - sql:disconnect` and then the second agent does the same, and so on. This means that when we open and close a connection in an **ask**-block we only use *one* connection at a time, independent of the number of agents in the agentset.

The example is more complex for **ask-concurrent**: the first agent performs `sql:connect - ... - set <agent-variable> <new value>`. As the state of an agent variable changed, the next agent is given a turn and performs the same commands and so on, finally the first agent continues with `... - sql:disconnect`. This means that connections are only closed after all agents have allocated connections, so here we use *as many* connections as there are agents.

For **ask-concurrent** this can mean that we exhaust the number of connections available on the database server (either because a limit is configured, or because the server simply can't handle more connections), especially when the number of agents is not known in advance.

2. Connectionpooling and autodisconnect

Connectionpooling simplifies database usage, because we only need to configure the `defaultconnection` once. With connectionpooling a connection is obtained when it is needed without the need for an explicit **sql:connect**. This simplicity however also introduces risk, because it is easy to forget that the connection needs to be closed as well. For that reason the connectionpool by default uses `autodisconnect`, this means that SQL Wrapper itself will close the connection as soon as possible:

- After changing the database (outside transactions)
- After reading the last result from a query (outside transactions)

- After committing or rolling back a transaction (see next section)

As an example where only the database is changed:

```
sql-configure "defaultconnection" <connection-list>

ask/ask-concurrent <agents> [
  sql:exec-update "INSERT INTO table (column) VALUES (?)" (list <agent variable>)
]
```

Here SQL Wrapper will only obtain a connection when **sql:exec-update** is executed. With autodisconnect enabled it will close that connection before the next command is executed. In this example it does not matter if we use **ask** or **ask-concurrent**, as we will use only *one* connection at a time. However if autodisconnect is disabled this example will open connections for every agent and those connections will not be closed! With autodisconnect off, we would need to add an explicit **sql:disconnect** to close the connection:

```
sql-configure  "defaultconnection [<connection-config> ["autodisconnect" "off"]]

ask/ask-concurrent <agents> [
  sql:exec-update "INSERT INTO table (column) VALUES (?)" (list <agent variable>)
  ...
  sql:disconnect
]
```

This looks simple enough, but now the difference in behaviour of **ask** and **ask-concurrent** can have an effect on the number of connections needed: if the `...` in the example does not change the state of the model (moving a turtle, changing global, turtle, link or patch variables) then **ask** and **ask-concurrent** will behave the same (and only *one* connection is needed). However if the state of the model is changed, in **ask-concurrent** this will switch the turn to the next agent (and so on) and we will need a connection for every agent. The connections are then only closed when all agents have had their turn for the first part of the command block.

When a query like **SELECT** is executed, the connection is only autodisconnected when the last result has been retrieved. This means that there are two potential causes for using too many connections: not reading all results (and not using an explicit **sql:disconnect**), or in **ask-concurrent** using commands or reporters that change the state of the model (and thus switch turn to the next agent).

For example:

```
sql-configure "defaultconnection" <connection-list>

ask/ask-concurrent <agents> [
  sql:exec-query "SELECT column1, column2 FROM table" []
  ...
  let row sql:fetch-row
  ...
]
```

In this example a connection is obtained in **sql:exec-query**. If the result of the query is only one row, then the connection will be released when sql:fetch-row is executed. If the result has more than one row (and we don't read them all), the connection will remain open. To prevent this, we either need to add an explicit sql:disconnect, or only query what we need, for example by adding `LIMIT 1` to the query to retrieve only one row. If we assume that autodisconnect will occur (because the query result is only one row), then the difference between **ask** and **ask-concurrent** once again depends on what happens at the first `...` as discussed earlier.

3. Transactions

The use of transactions adds yet another layer of complexity when autodisconnect is on, as the autodisconnect is suspended until transaction commit or rollback. For the purpose of this discussion we can ignore the case when autodisconnect is off, as the consumption of connections is similar to the cases already discussed. This section focuses on the impact of transactions on connection usage, the next chapter, *Using transactions*, discusses the use of transactions itself.

The basic flow of a transaction looks like this:

```
sql-configure "defaultconnection" <connection-list>

ask/ask-concurrent <agents> [
  sql:start-transaction
  ...
  sql:exec-update "INSERT INTO table (column) VALUES (?)" (list <agent variable>)
  ...
  sql:commit-transaction
]
```

Contrary to previous examples, the connection is now obtained from the connectionpool at **start-transaction**, and the connection is not closed by **sql:exec-update**, but only when **commit-transaction** is executed (assuming autodisconnect is on). Once again for **ask** this still means only one connection is needed, but for **ask-concurrent** it depends if the first and/or second . . . changes the state of the model.

4. Minimizing connection use

In the previous sections we discussed how the number of connections is impacted by aspects of the NetLogo language and SQL Wrapper itself. Here we list the options to limit the number of connections used:

- Use connectionpooling with autodisconnect
- Use **sql:disconnect** as soon as possible (when creating connections using **sql:connect** or connectionpooling with autodisconnect off)
- Use **ask** instead of **ask-concurrent** when possible
- With **ask-concurrent**: Do not change the state of the model until after the connection is closed (eg save information to local variables and assign to global, turtle, link or patch variables after closing the connection)
- Query only the number of rows that is needed (eg use `LIMIT` in the query), or use an explicit **sql:disconnect**
- Only use transactions when needed and make them as short as possible

The NetLogo language also has a command **without-interruption** that can be used to execute a block of commands in a **ask-concurrent** block without switching the turn to another agent. Use of this command is advisable when you do need or want to use **ask-concurrent**, but don't need to interleave the commands using the database connection.

5. Configuring connections

By default the connectionpool will create up to 20 connections. If SQL Wrapper tries to obtain a 21st connection, this will - by default - throw an error. If the timeout on the Section 1.2, “Aspect: "connectionpool"” is disabled it will block until a connection is available. Unfortunately due to NetLogo's single threaded approach a connection will never come available during this blocking call so it is advisable to keep the timeout set!

If you follow the strategies described in the previous sections, you will usually only need 1 or 2 connections at a time, but for more complex NetLogo models it may be necessary to keep multiple connections open at once. If this exceeds the limit of 20 connections, you will need to increase the `maxconnections` option of Section 1.2, “Aspect: "connectionpool"”. Configuring this option needs to happen before configuring the connection itself:

```
sql:configure "connectionpool" [["maxconnections" 30]]
sql:configure "defaultconnection" <connection-list>
```

Important

The number of connections can also be limited by one or more database settings, please consult with your database administrator to find out if limits apply and to increase limits if necessary.

Chapter 12. Using transactions

In the default use of SQL Wrapper (without transactions), every statement is executed independently and technically: in its own transaction. You use transaction when you need a group of statements to be completed as one (or to be undone as one), or when you need to have a consistent view of the database (you don't want to see the changes that occurred after you started querying the database). Transactions also ensure that two different agents cannot change the same item of data at once.

Important

Transactions in MySQL only work for InnoDB tables, please ensure that all your tables use the InnoDB-engine if you want to use transactions!

Use of transactions usually implies that other users of the database cannot see your changes until you commit them. This visibility depends on the transaction isolation level used by the other user. SQL wrapper uses the REPEATABLE-READ isolation level: you can only see information from transactions committed before you started querying the database (but changes created by the own transaction are visible). The current version of SQL Wrapper has no option to change this isolation level.

In general use of transactions look like this:

```
sql:configure "defaultconnection" <connection-list>

ask <agents> [
  sql:start-transaction
  ...
  sql:commit-transaction
]
```

or:

```
sql:configure "defaultconnection" <connection-list>

ask <agents> [
  sql:start-transaction
  ...
  ifelse <condition>
    [sql:commit-transaction]
    [sql:rollback-transaction]
]
```

Warning

The transaction behavior of the current version of SQL Wrapper is unspecified if an error occurs when executing a query.

Chapter 13. Connecting to other databases

The generic brand option in **sql:connect** and Connection pool (or automatic connection management) enable you to connect to databases not explicitly supported by SQL Wrapper, or to specify advanced connection properties not exposed for the supported databases. SQL Wrapper uses Java DataBase Connectivity (JDBC) to connect to these databases. Almost all database systems have support for JDBC (in the form of JDBC drivers).

To use the generic brand option, you will first need to obtain the JDBC driver for your database. This driver is usually downloadable from the website of the database-vendor. Next you need to read the documentation of the JDBC driver to find out the following things:

1. The filename(s) of the driver-archive(s) (jar-file)

2. The name of the driver-class

3. The syntax and rules for defining a JDBC-url to connect to your database

To install the driver, copy the driver-archive file(s) into the `extensions/sql` directory inside the NetLogo installation directory (this directory also contains the other SQL Wrapper files). After installing the driver files, you will need to restart NetLogo to be able to use the drivers.

Warning

If you update to a newer version of a JDBC driver, make sure you delete the old driver, otherwise it is possible that the old driver will still be used.

Next, find out the correct JDBC-url for your database using the documentation or support of your database vendor. Every database vendor has its own syntax-rules, so carefully read the documentation for your database. Some databases allow you to include the username and password into the JDBC-url. SQL Wrapper however expects you to set the username and password separately.

With the information gathered you should now be able to connect to your database using:

```
sql:connect [["brand" "generic"] ["driver" "<name of your driver>"] ["jdbc-url" "<your
 JDBC-url>"] ["user" "<your username>"] ["password" "<your password>"]]
```

In this way you can connect to a large number of databases, including Oracle, Microsoft SQL Server and Firebird.

Important

The JDBC driver should be a JDBC Type-4 (or pure-Java) driver and not require loading any external native library file (.dll or .so) to connect to the database.

1. Connecting to an Oracle database

Download the JDBC driver from Oracle (see http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html). The file you need depends on the Java version you are using, for Java 6 you need ojdbc6.jar . This file should go into the extensions/sql directory. According to Oracle, this driver will support connections with the following versions of Oracle database servers: 11.2.0, 11.1.0, 10.2.0, 10.1.0, 9.2.0, 9.01. We tested this with Oracle 10.2.0g Express Edition.

For establishing the database connection, either through **sql:connect** or through configuring the default connections, your jdbc-url and driver parameters should include the following parameters:

```
["brand" "generic"] ["jdbc-url" "jdbc:oracle:thin:@localhost:1521:xe"] ["driver"
 "oracle.jdbc.OracleDriver"]
```

where "xe" at the end of the jdbc-url string identifies the Oracle SID you will be using.

Note

You must be using TCP/IP for the database connection. OCI is not supported. Also note that if you use a "light" or "express edition" of Oracle, you might find that the process limits of the server are too limited. The following link provides information on how this limit can be increased: http://www.atpeaz.com/index.php/2010/fixing-the-ora-12519-tnsno-appropriate-service-handler-found-error/

In some configuration of Oracle, Java and NetLogo, you might receive an error 'OAUTH marshalling failed'. To workaround this error message, you will need to edit the file '`NetLogo 4.1.3.vmoptions`' in the NetLogo directory and add the following line to it (before the line containing the text '`null`'):

```
-Doracle.jdbc.thinLogonCapability=o3
```

Other than the information above, we are unable to provide information or assistance with regards to connecting to Oracle databases.

Appendix A. GNU Lesser General Public License version 3

Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a. under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b. under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a. Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b. Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a. Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b. Accompany the Combined Work with a copy of the GNU GPL and this license document.

c. For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d. Do one of the following:

    1. Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

    2. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e. Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b. Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Appendix B. GNU General Public License version 3

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
one line to give the program’s name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
```

```
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see http://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.