ceODBC

Release 3.1.0

August 01, 2023

Contents

ceODBC is a Python extension module that enables access to databases using the ODBC API and conforms to the Python database API 2.0 specifications with a number of additions. See https://www.python.org/dev/peps/pep-0249 for more information on the Python database API specification.

ceODBC is distributed under an open-source *license* (the PSF license).

Contents:

CHAPTER 1

Module Interface

ceODBC.**Binary**(*string*)
Construct an object holding a binary (long) string value. This is merely a wrapper over the bytes class and that should be used instead.

ceODBC.**Connection**(*dsn*, *autocommit=False*)
ceODBC.**connect**(*dsn*, *autocommit=False*)
Constructor for creating a *connection*. The only required argument is the DSN in the format that ODBC expects. The autocommit flag can be set in the constructor or it can be manipulated after the connection has been established. If you are using a driver that does not handle transactions, ensure that this value is set to True or you may get a "driver not capable" exception.

ceODBC.**Cursor**(*connection*)
Constructor for creating a *cursor* using the connection.

Note: This method is an extension to the DB API definition.

---

ceODBC.**data_sources**(*exclude_user_dsn=False*, *exclude_system_dsn=False*)
Return a list of 2-tuples identifying the configured data sources. By default all data sources are returned but system or user data sources can be excluded, if desired.

Note: This method is an extension to the DB API definition.

---

ceODBC.**Date**(*year*, *month*, *day*)
Construct an object holding a date value. This is merely a wrapper over the datetime.date class and that should be used instead.

ceODBC.**DateFromTicks**(*ticks*)
Construct an object holding a date value from the given ticks value (number of seconds since the epoch; see the documentation of the standard Python time module for details). This is equivalent to using datetime.date.fromtimestamp() and that should be used instead.

ceODBC.**drivers**()
Return a list of the names of the configured drivers.

Note: This method is an extension to the DB API definition.

ceODBC.**Time**(*hour*, *minute*, *second*)
Construct an object holding a time value. This is merely a wrapper over the datetime.time class and that should be used instead.

ceODBC.**TimeFromTicks**(*ticks*)
Construct an object holding a time value from the given ticks value (number of seconds since the epoch; see the documentation of the standard Python time module for details). This is equivalent to using datetime.datetime.fromtimestamp().time() and that should be used instead.

ceODBC.**Timestamp**(*year*, *month*, *day*, *hour*, *minute*, *second*)
Construct an object holding a time stamp value. This is merely a wrapper over the datetime.datetime class and that should be used instead.

ceODBC.**TimestampFromTicks**(*ticks*)
Construct an object holding a time stamp value from the given ticks value (number of seconds since the epoch; see the documentation of the standard Python time module for details). This is equivalent to using datetime.datetime.fromtimestamp() and that should be used instead.

1.1 Constants

ceODBC.**apilevel**
String constant stating the supported DB API level. Currently '2.0'.

ceODBC.**BINARY**
This type object is used to describe columns in a database that are binary.

ceODBC.**DATETIME**
This type object is used to describe columns in a database that are dates.

ceODBC.**NUMBER**
This type object is used to describe columns in a database that are numbers.

ceODBC.**paramstyle**
String constant stating the type of parameter marker formatting expected by the interface. Currently 'qmark' as in 'where name = ?'.

ceODBC.**ROWID**
This type object is used to describe the pseudo column "rowid".

ceODBC.**STRING**
This type object is used to describe columns in a database that are strings.

ceODBC.**threadsafety**
Integer constant stating the level of thread safety that the interface supports. Currently 2, which means that threads may share the module and connections, but not cursors. Sharing means that a thread may use a resource without wrapping it using a mutex semaphore to implement resource locking.

ceODBC.**__version__**
String constant stating the version of the module. Currently '3.1.0'.

Note: This attribute is an extension to the DB API definition.

1.2 Exceptions

exception ceODBC.**Warning**
Exception raised for important warnings and defined by the DB API but not actually used by ceODBC.

exception ceODBC.**Error**
Exception that is the base class of all other exceptions defined by ceODBC and is a subclass of the Python StandardError exception (defined in the module exceptions).

exception ceODBC.**InterfaceError**
Exception raised for errors that are related to the database interface rather than the database itself. It is a subclass of Error.

exception ceODBC.**DatabaseError**
Exception raised for errors that are related to the database. It is a subclass of Error.

exception ceODBC.**DataError**
Exception raised for errors that are due to problems with the processed data. It is a subclass of DatabaseError.

exception ceODBC.**OperationalError**
Exception raised for errors that are related to the operation of the database but are not necessarily under the control of the progammer. It is a subclass of DatabaseError.

exception ceODBC.**IntegrityError**
Exception raised when the relational integrity of the database is affected. It is a subclass of DatabaseError.

exception ceODBC.**InternalError**
Exception raised when the database encounters an internal error. It is a subclass of DatabaseError.

exception ceODBC.**ProgrammingError**
Exception raised for programming errors. It is a subclass of DatabaseError.

exception ceODBC.**NotSupportedError**
Exception raised when a method or database API was used which is not supported by the database. It is a subclass of DatabaseError.

1.3 Database Types

Note: The DB API definition does not define these objects.

These types are more granular than the types mandated by the DB API and can be used when creating variables via *Cursor.var()* or *Cursor.setinputsizes()*.

ceODBC.**DB_TYPE_BIGINT**
Variable used to bind and/or fetch big integers. Values are returned as Python integers and accept the same.

ceODBC.**DB_TYPE_BINARY**
Variable used to bind and/or fetch binary data. Values are returned as Python bytes objects and accept the same.

ceODBC.**DB_TYPE_BIT**
Variable used to bind and/or fetch bits. Values are returned as Python booleans and accept the same.

ceODBC.**DB_TYPE_DATE**
Variable used to bind and/or fetch dates. Values are returned as Python datetime.date objects and accept Python datetime.date or datetime.datetime objects.

ceODBC.**DB_TYPE_DECIMAL**
Variable used to bind and/or fetch decimal numbers. Values are returned as Python decimal.Decimal objects and accept the same.

ceODBC.**DB_TYPE_DOUBLE**
Variable used to bind and/or fetch floating point numbers. Values are returned as Python floats and accept Python integers or floats.

ceODBC.**DB_TYPE_INT**
Variable used to bind and/or fetch integers. Values are returned as Python integers and accept the same.

ceODBC.**DB_TYPE_LONG_BINARY**
Variable used to bind and/or fetch long binary data. Values are returned as Python bytes objects and accept the same.

ceODBC.**DB_TYPE_LONG_STRING**
Variable used to bind and/or fetch long string data. Values are returned as Python strings and accept the same.

ceODBC.**DB_TYPE_STRING**
Variable used to bind and/or fetch string data. Values are returned as Python strings and accept the same.

ceODBC.**DB_TYPE_TIME**
Variable used to bind and/or fetch time data. Values are returned as Python datetime.time objects and accept Python datetime.time or datetime.datetime objects.

ceODBC.**DB_TYPE_TIMESTAMP**
Variable used to bind and/or fetch timestamps. Values are returned as Python datetime.datetime objects and accept Python datetime.date or datetime.datetime objects.

CHAPTER 2

Connection Object

Note: Any outstanding changes will be rolled back when the connection object is destroyed or closed.

`Connection.__enter__()`
The entry point for the connection as a context manager.

Note: This method is an extension to the DB API definition.

`Connection.__exit__()`
The exit point for the connection as a context manager. In the event of an exception, the transaction is rolled back; otherwise, the transaction is committed.

Note: This method is an extension to the DB API definition.

`Connection.autocommit`
This read-write attribute returns the setting of the autocommit flag for the connection. When set, any statements executed are automatically committed if successful; otherwise, a commit() or rollback() must be issued for the changes to be committed to (or rolled back from) the database.

Note: This attribute is an extension to the DB API definition.

`Connection.close()`
Close the connection now, rather than whenever __del__ is called. The connection will be unusable from this point forward; an Error exception will be raised if any operation is attempted with the connection. The same applies to any cursor objects trying to use the connection.

`Connection.columnprivileges`(*catalog=None*, *schema=None*, *table=None*, *column=None*)
Return a cursor containing the privileges for columns in the catalog filtered by the parameters catalog, schema, table and column as desired. See the ODBC API reference for SQLColumnPrivileges() for more information.

Note: This method is an extension to the DB API definition.

Connection.**columns**(*catalog=None*, *schema=None*, *table=None*, *column=None*)
Return a cursor containing the columns in the catalog filtered by the parameters catalog, schema, table and column as desired. See the ODBC API reference for SQLColumns() for more information.

Note: This method is an extension to the DB API definition.

Connection.**commit**()
Commit any pending transactions to the database.

Connection.**cursor**()
Return a new Cursor object (*Cursor Object*) using the connection.

Connection.**dsn**
This read-only attribute returns the DSN of the database to which a connection has been established.

Note: This attribute is an extension to the DB API definition.

Connection.**inputtypehandler**
This read-write attribute specifies a method called for each value that is bound to a statement executed on any cursor associated with this connection, unless a different handler is specified for that cursor. The method signature is handler(cursor, value, arraysize) and the return value is expected to be a variable object or None in which case a default variable object will be created. If this attribute is None, the default behavior will take place for all values bound to statements.

Note: This attribute is an extension to the DB API definition.

Connection.**foreignkeys**(*pkcatalog=None*, *pkschema=None*, *pktable=None*, *fkcatalog=None*, *fkschema=None*, *fktable=None*)
Return a cursor containing the foreign keys in the catalog filtered by the parameters catalog, schema and table for both the primary and foreign key table as desired. See the ODBC API reference for SQLForeignKeys() for more information.

Note: This method is an extension to the DB API definition.

Connection.**outputtypehandler**
This read-write attribute specifies a method called for each value that is to be fetched from any cursor associated with this connection, unless a different handler is specified for that cursor. The method signature is handler(cursor, name, defaultType, length, scale) and the return value is expected to be a variable object or None in which case a default variable object will be created. If this attribute is None, the default behavior will take place for all values fetched from cursors.

Note: This attribute is an extension to the DB API definition.

Connection.**primarykeys**(*catalog=None*, *schema=None*, *table=None*)
Return a cursor containing the primary key columns in the catalog filtered by the parameters catalog, schema and table as desired. See the ODBC API reference for SQLPrimaryKeys() for more information.

Note: This method is an extension to the DB API definition.

Connection.**procedurecolumns**(*catalog=None*, *schema=None*, *proc=None*, *column=None*)
Return a cursor containing the columns for procedures in the catalog filtered by the parameters catalog, schema, proc and column as desired. See the ODBC API reference for SQLProcedureColumns() for more information.

Note: This method is an extension to the DB API definition.

Connection.**procedures**(*catalog=None*, *schema=None*, *proc=None*)
Return a cursor containing the procedures in the catalog filtered by the parameters catalog, schema and proc as desired. See the ODBC API reference for SQLProcedures() for more information.

Note: This method is an extension to the DB API definition.

Connection.**rollback**()
Rollback any pending transactions.

Connection.**tableprivileges**(*catalog=None*, *schema=None*, *table=None*)
Return a cursor containing the privileges for tables in the catalog filtered by the parameters catalog, schema and table as desired. See the ODBC API reference for SQLTablePrivileges() for more information.

Note: This method is an extension to the DB API definition.

Connection.**tables**(*catalog=None*, *schema=None*, *table=None*)
Return a cursor containing the tables in the catalog filtered by the parameters catalog, schema and table as desired. See the ODBC API reference for SQLTables() for more information.

Note: This method is an extension to the DB API definition.

CHAPTER 3

Cursor Object

Cursor.**arraysize**
: This read-write attribute specifies the number of rows to fetch at a time internally and is the default number of rows to fetch with the *fetchmany()* call. It defaults to 1 meaning to fetch a single row at a time. Note that this attribute can drastically affect the performance of a query since it directly affects the number of network round trips that need to be performed.

Cursor.**bindarraysize**
: This read-write attribute specifies the number of rows to bind at a time and is used when creating variables via *setinputsizes()*. It defaults to 1 meaning to bind a single row at a time.

Note: The DB API definition does not define this attribute.

Cursor.**callfunc**(*name*, *return_type*, * *args*)
: Call a function with the given name. Parameters may also be passed as a single list or tuple to conform to the DB API. The return type is specified in the same notation as is required by *setinputsizes()*. The result of the call is the return value of the function.

Note: The DB API definition does not define this method.

Cursor.**callproc**(*name*, * *args*)
: Call a procedure with the given name. Parameters may also be passed as a single list or tuple to conform to the DB API. The result of the call is a modified copy of the input sequence. Input parameters are left untouched; output and input/output parameters are replaced with possibly new values.

Cursor.**close**()
: Close the cursor now, rather than whenever __del__ is called. The cursor will be unusable from this point forward; an Error exception will be raised if any operation is attempted with the cursor.

Cursor.**connection**
: This read-only attribute returns a reference to the connection object on which the cursor was created.

Note: This attribute is an extension to the DB API definition but it is mentioned in PEP 249 as an optional extension.

`Cursor.`**`description`**

This read-only attribute is a sequence of 7-item sequences. Each of these sequences contains information describing one result column: (name, type, display_size, internal_size, precision, scale, null_ok). This attribute will be None for operations that do not return rows or if the cursor has not had an operation invoked via the execute() method yet.

The type will be one of the database type objects (*Database Types*) and is comparable to the type objects defined by the DB API.

`Cursor.`**`execdirect`**(*statement*)

Execute a statement against the database using SQLExecDirect instead of SQLExecute. This is necessary in some situations due to bugs in ODBC drivers such as exhibited by the SQL Server ODBC driver when calling certain stored procedures.

If the statement is a query, the cursor is returned as a convenience since cursors implement the iterator protocol and there is thus no need to call one of the appropriate fetch methods; otherwise None is returned.

Note: The DB API definition does not define this method.

`Cursor.`**`execute`**(*statement*, ** args*)

Execute a statement against the database. Paramters may also be passed as a single list or tuple to conform to the DB API.

A reference to the statement will be retained by the cursor. If None or the same string object is passed in again, the cursor will execute that statement again without performing a prepare or rebinding and redefining. This is most effective for algorithms where the same statement is used, but different parameters are bound to it (many times).

For maximum efficiency when reusing an statement, it is best to use the *`setinputsizes()`* method to specify the parameter types and sizes ahead of time; in particular, None is assumed to be a string of length 1 so any values that are later bound as numbers or dates will raise a TypeError exception.

If the statement is a query, the cursor is returned as a convenience since cursors implement the iterator protocol and there is thus no need to call one of the appropriate fetch methods; otherwise None is returned.

Note: The DB API definition does not define the return value of this method.

`Cursor.`**`executemany`**(*statement*, *parameters*)

Prepare a statement for execution against a database and then execute it against all parameter sequences found in the sequence parameters. The statement is managed in the same way as the *`execute()`* method manages it.

`Cursor.`**`fetchall`**()

Fetch all (remaining) rows of a query result, returning them as a list of tuples. An empty list is returned if no more rows are available. Note that the cursor's arraysize attribute can affect the performance of this operation, as internally reads from the database are done in batches corresponding to the arraysize.

An exception is raised if the previous call to execute() did not produce any result set or no call was issued yet.

`Cursor.`**`fetchmany`**([*num_rows=cursor.arraysize*])

Fetch the next set of rows of a query result, returning a list of tuples. An empty list is returned if no more rows are available. Note that the cursor's arraysize attribute can affect the performance of this operation.

The number of rows to fetch is specified by the parameter. If it is not given, the cursor's arrysize attribute determines the number of rows to be fetched. If the number of rows available to be fetched is fewer than the amount requested, fewer rows will be returned.

An exception is raised if the previous call to execute() did not produce any result set or no call was issued yet.

Cursor.**fetchone**()
Fetch the next row of a query result set, returning a single tuple or None when no more data is available.

An exception is raised if the previous call to execute() did not produce any result set or no call was issued yet.

Cursor.**inputtypehandler**
This read-write attribute specifies a method called for each value that is bound to a statement executed by this cursor, and overrides the attribute with the same name on the connection if specified. The method signature is handler(cursor, value, arraysize) and the return value is expected to be a variable object or None in which case a default variable object will be created. If this attribute is None, the value of the attribute with the same name on the connection is used.

Note: This attribute is an extension to the DB API definition.

Cursor.**__iter__**()
Returns the cursor itself to be used as an iterator.

Note: This method is an extension to the DB API definition but it is mentioned in PEP 249 as an optional extension.

Cursor.**name**
This read-write attribute returns the name associated with the cursor. This name is used in positioned update or delete statements (as in delete from X where current of <NAME>).

Note: This attribute is an extension to the DB API definition.

Cursor.**next**()
Fetch the next row of a query result set, using the same semantics as the method fetchone().

Note: This method is an extension to the DB API definition but it is mentioned in PEP 249 as an optional extension.

Cursor.**nextset**()
Make the cursor skip to the next available set, discarding any remaining row from the current set. If there are no more sets, None is returned; otherwise, the cursor itself is returned as a convenience for fetching data from it. Note that not all databases support the concept of multiple result sets.

Cursor.**outputtypehandler**
This read-write attribute specifies a method called for each value that is to be fetched from this cursor and overrides the attribute with the same name on the connection if specified. The method signature is handler(cursor, name, defaultType, length, scale) and the return value is expected to be a variable object or None in which case a default variable object will be created. If this attribute is None, the value of the attribute with the same name on the connection is used.

Note: This attribute is an extension to the DB API definition.

Cursor.**prepare**(*statement*)
This can be used before a call to execute() to define the statement that will be executed. When this is done, the prepare phase will not be performed when the call to execute() is made with None or the same string object as the statement.

Note: The DB API definition does not define this method.

Cursor.**rowcount**
This read-only attribute specifies the number of rows that have currently been fetched from the cursor (for select statements) or that have been affected by the operation (for insert, update and delete statements).

Cursor.**rowfactory**
This read-write attribute specifies a method to call for each row that is retrieved from the database. Ordinarily a tuple is returned for each row but if this attribute is set, the method is called with the argument tuple that would normally be returned and the result of the method is returned instead.

Note: The DB API definition does not define this attribute.

Cursor.**setinputsizes**(** args*)
This can be used before a call to execute() to predefine memory areas for the operation's parameters. Each parameter should be a type object corresponding to the input that will be used or it should be an integer specifying the maximum length of a string parameter. The singleton None can be used as a parameter to indicate that no space should be reserved for that position. Note that in order to conform to the DB API, passing a single argument which is a list or tuple will treat that list or tuple as the arguments sequence.

Cursor.**setoutputsize**(*size*[, *column*])
This can be used before a call to execute() to predefine memory areas for the long columns that will be fetched. The column is specified as an index into the result sequence. Not specifying the column will set the default size for all large columns in the cursor.

Cursor.**statement**
This read-only attribute provides the string object that was previously prepared with prepare() or executed with execute().

Note: The DB API definition does not define this attribute.

Cursor.**var**(*type*, *size=0*, *scale=0*, *arraysize=1*, *inconverter=None*, *outconverter=None*, *input=True*, *output=False*)
Create a variable associated with the cursor of the given type and characteristics and return a variable object (*Variable Objects*). If the arraysize is not specified, the bind array size (usually 1) is used. The inconverter and outconverter specify methods used for converting values to/from the database. More information can be found in the section on variable objects.

This method was designed for use with in/out variables where the length or type cannot be determined automatically from the Python object passed in or for use in input and output type handlers defined on cursors or connections.

Note: The DB API definition does not define this method.

CHAPTER 4

Variable Objects

Note: The DB API definition does not define this object.

`Variable.`**`buffer_size`**
This read-only attribute returns the size of the buffer allocated for each element.

`Variable.`**`getvalue`**`(`[*pos=0*]`)`
Return the value at the given position in the variable.

`Variable.`**`inconverter`**
This read-write attribute specifies the method used to convert data from Python to the database. The method signature is converter(value) and the expected return value is the value to bind to the database. If this attribute is None, the value is bound directly without any conversion.

`Variable.`**`input`**
This read-write attribute specifies whether the variable is used as an input variable and should normally be left as True.

`Variable.`**`num_elements`**
This read-only attribute returns the number of elements allocated.

`Variable.`**`outconverter`**
This read-write attribute specifies the method used to convert data from from the database to Python. The method signature is converter(value) and the expected return value is the value to return to Python. If this attribute is None, the value is returned directly without any conversion.

`Variable.`**`output`**
This read-write attribute specifies whether the variable is used as an output variable. It should normally be left as False except when calling stored procedures with output variables.

`Variable.`**`scale`**
This read-only attribute returns the scale of the variable.

`Variable.`**`setvalue`**`(`*pos*, *value*`)`
Set the value at the given position in the variable.

`Variable.`**`size`**
: This read-only attribute returns the size of the variable.

`Variable.`**`type`**
: This read-only attribute returns the type of the variable.

CHAPTER 5

Release notes

5.1 3.x releases

5.1.1 Version 3.1

1) Dropped support for Python 3.6. Support is now for Python 3.7 and higher.
2) Added support for listing the available data sources and drivers (issue 8).
3) Added support for the CHAR data type (issue 9).
4) Restored support for boolean columns.
5) Adjust builds to use pyproject.toml exclusively. Binary packages are built using cibuildwheel.

5.1.2 Version 3.0

1) Dropped support for Python 2. Support is now for Python 3.6 and higher.
2) Migrated module to a Python package with the use of Cython for speedups.
3) Migrated test suite to using tox in order to automate testing of different environments.
4) Added input and output type handlers on cursors and connections. This enables the default types to be overridden if desired. See the documentation for more details.
5) Added better support for 64-bit Python.
6) Eliminated compiler warnings; other minor tweaks to improve error handling.
7) Dropped attribute *ceODBC.buildtime*.
8) Dropped use of cx_Logging for logging.

5.2 Older versions

5.2.1 Version 2.0.1

1) Removed memory leak that occurred when binding parameters to a cursor; thanks to Robert Ritchie and Don Reid for discovering this.
2) Remove the password from the DSN in order to eliminate potential security leaks.
3) Improve performance when logging is disabled or not at level DEBUG by avoiding the entire attempt to log bind variable values.
4) Use the size value rather than the length value when defining result set variables since the length value is for the length of the column name; thanks to Heran Quan for the patch.
5) Added support for Python 3.2.

5.2.2 Version 2.0

1) Added support for Python 3.x and Unicode.
2) Added support for 64-bit Python installations.
3) Added test suites for MySQL, PostgreSQL and SQL Server.
4) Added support for cursor nextset().
5) Added support for cursor execdirect() which calls SQLExecDirect() instead of SQLExecute() which can be necessary in order to work around bugs in various ODBC drivers.
6) Added support for creating variables and for specifying input and output converters as in cx_Oracle.
7) Added support for deferred type assignment for cursor executemany() as in cx_Oracle.
8) Fixed a number of bugs found by testing against various ODBC drivers.

5.2.3 Version 1.2

1) Added support for time data as requested by Dmitry Solitsky.
2) Added support for Python 2.4 as requested by Lukasz Szybalski.
3) Added support for setting the autocommit flag in the connection constructor since some drivers do not support transactions and raise a "driver not capable" exception if any attempt is made to turn autocommit off; thanks to Carl Karsten for working with me to resolve this problem.
4) Added support for calculating the size and display size of columns in the description attribute of cursors as requested by Carl Karsten.
5) Use SQLFreeHandle() rather than SQLCloseCursor() since closing a cursor in the ODBC sense is not the same as closing a cursor in the DB API sense and caused strange exceptions to occur if no query was executed before calling cursor.close().
6) Added additional documentation to README.txt as requested by Lukasz Szybalski.
7) Tweaked setup script and associated configuration files to make it easier to build and distribute; better support for building with cx_Logging if desired.

5.2.4 Version 1.1

1) Added support for determining the columns, column privileges, foreign keys, primary keys, procedures, procedure columns, tables and table privileges available in the catalog as requested by Dmitry Selitsky.

2) Added support for getting/setting the autocommit flag for connections.

3) Added support for getting/setting the cursor name which is useful for performing positioned updates and deletes (as in delete from X where current of cursorname).

4) Explicitly set end of rows when SQL_NO_DATA is returned from SQLFetch() as some drivers do not properly set the number of rows fetched.

CHAPTER 6

License

LICENSE AGREEMENT FOR ceODBC

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the disclaimer that follows.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the names of the copyright holders nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS *AS IS* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

B

C

D

E

F

G

I

N

O

P

R

S

T

V

W