BCP

Release 0.4.0

Feb 03, 2020

Contents:

The source repository is located here

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

CHAPTER 2

User-Facing Classes

2.1 BCP

This is the core module of the library, containing the primary interface over the functionality of bcp. Along with its dependencies on Connection and DataFile, it serves as the entry point to the library. Simply create a Connection, pass it in to the BCP object, and then use load() or dump() to read data into and out of a database. See the methods below for examples.

class bcp.core.**BCP**(*connection*)

This is the interface over dialect-specific classes that provides generic methods to load/dump data to/from a database.

Parameters **connection** (Connection) – a Connection object that contains authorization details and database details

Example:

```
import bcp

conn = bcp.Connection('host', 'mssql', 'username', 'password')
my_bcp = bcp.BCP(conn)
```

dump(*query*, *output_file*)

This method provides an interface to the lower level dialect-specific BCP dump classes.

Parameters

- **query** (str) – the query whose results should be saved off to a file
- **output_file** (DataFile) – the file to which the data should be saved, if no file is provided, one will be created in the BCP_DATA_DIR

Example:

```
import bcp
```

(continues on next page)

(continued from previous page)

```
conn = bcp.Connection(host='HOST', driver='mssql', username='USER', password=
↪'PASSWORD')
my_bcp = bcp.BCP(conn)
file = bcp.DataFile(file_path='path/to/file.csv', delimiter=',')
my_bcp.dump(query='select * from sys.tables', output_file=file)
```

load(*input_file*, *table*)

This method provides an interface to the lower level dialect-specific BCP load classes

Parameters

- **input_file** (DataFile) – the file to be loaded into the database
- **table** (str) – the table in which to land the data

Example:

```
import bcp

conn = bcp.Connection(host='HOST', driver='mssql', username='USER', password=
↪'PASSWORD')
my_bcp = bcp.BCP(conn)
file = bcp.DataFile(file_path='path/to/file.csv', delimiter=',')
my_bcp.load(input_file=file, table='table_name')
```

2.2 Connections

This module contains data structures required to connect to a database. While Auth can be instantiated on its own, you would generally create it at the same time as the connection. Since the Auth object will contain credentials, it's recommended to set these through secure means, such as environmental variables, so as to not accidentally check in your credentials to your source control. Even your host name can be considered sensitive data, depending on your and your company's policies.

Example:

```
import os
import bcp

host = os.environ['HOST']
username = os.environ['USERNAME']
password = os.environ['PASSWORD']

conn = bcp.Connection(host, 'mssql', username, password)
my_bcp = bcp.BCP(conn)
```

class bcp.connections.**Auth**(*username=None*, *password=None*)

This data structure collects the username and password as an authentication object.

Parameters

- **username** (Optional[str]) – username for the authorization
- **password** (Optional[str]) – password for the authorization

type

This property identifies the authorization type depending on the username/password provided. The two options for authorization are Trusted and Credential. A Trusted connection is created when no username

and no password are provided. In this case, the local user's credentials and authorization method are used. A Credential connection is created when both a username and a password are provided. If only one of username and password are provided, this raises an InvalidCredentialException.

Return type str

Returns the type of connection ('Trusted' or 'Credential')

class bcp.connections.**Connection**(*driver*, *host*, *port=None*, *username=None*, *password=None*)

This data structure describes a connection to be used to instantiate a BCP instance. A host and driver must be supplied. A username/password combination can also be supplied upon instantiation to automatically create an associated Auth object. Alternatively, this can be set as an attribute after instantiation. If the username/password are not provided, the connection will assume a Trusted authorization in the meantime.

Parameters

- **driver** (str) – the type of database (mssql, etc.)
- **host** (str) – the host where the database exists
- **port** (Optional[int]) – the port for the database server
- **username** (Optional[str]) – the username for authentication
- **password** (Optional[str]) – the password for authentication

2.3 Files

This module contains data structures required to create and access files. Users will generally only need to use DataFile directly. LogFile and ErrorFile are used indirectly by the BCP classes.

Example:

```
from bcp import DataFile

# create a csv to write out my data
my_file = DataFile(delimiter=',')
print(my_file.path)  # %HOME%/bcp/data/<timestamp>.csv
```

class bcp.files.**DataFile**(*file_path=None*, *delimiter=None*)

This is a handle to a data file.

Parameters

- **file_path** (Optional[Path]) – the path object to the file, if not provided, a default using the current timestamp will be created
- **delimiter** (Optional[str]) – the field delimiter for the data file

class bcp.files.**ErrorFile**(*file_path=None*)

This is a handle to an error file.

Parameters **file_path** (Optional[Path]) – the path object to the file, if not provided, a default using the current timestamp will be created

class bcp.files.**File**

This data structure creates a file handle given a file path. If the file path is not provided:

- the current timestamp is used so that unique error, log, and data files can be created
- the file will be created in the BCP_ROOT_DIR directory specified in config.py

class bcp.files.**LogFile**(*file_path=None*)
This is a handle to a log file.

Parameters **file_path** (Optional[Path]) – the path object to the file, if not provided, a default using the current timestamp will be created

CHAPTER 3

Dialect Support

class bcp.dialects.base.**BCPDump**(*connection*, *query*, *file*)

This is the abstract base class for all driver specific Dump implementations. It contains required methods and default values for all subclasses.

Parameters

- **connection** (Connection) – the Connection object that points to the database from which we want to export data
- **query** (str) – the query for the data to be exported
- **file** (DataFile) – the file to write the to, if not provided, a default will be created in the BCP_DATA_DIR

execute()

This will execute the data export process

Returns the data file object, which is useful when it is defaulted

class bcp.dialects.base.**BCPLoad**(*connection*, *file*, *table*)

This is the abstract base class for all dialect-specific Dump implementations. It contains required methods and default values for all subclasses.

Parameters

- **connection** (Connection) – the Connection object that points to the database from which we want to export data
- **file** (DataFile) – the file to be loaded into the database
- **table** (str) – the table in which to land the data

execute()

This will execute the the data import process.

Returns the name of the table that contains the data

3.1 MSSQL

This module contains MS SQL Server specific logic that works with the BCP command line utility. None of these classes are relevant beyond the scope of this library, and should not be used outside of the library.

class bcp.dialects.mssql.**MSSQLBCP**
This mixin provides properties for MSSQL's BCP. It serves as a mixin for BCPLoad and BCPDump subclasses below.

command
This allows you to see the command that will be executed via bcp without executing the command, similar to how sqlalchemy can show you the generated query or execute the generated query. This makes debugging a little easier.

Return type str

Returns the command that will be passed into the bcp command line utility

config
This method will generate a configuration string. It supports the delimiter option.

Return type str

Returns a BCP formatted configuration string

logging
This method will generate a log file string that will write the log file to the BCP_LOGGING_DIR directory.

Return type str

Returns a BCP formatted log file string

class bcp.dialects.mssql.**MSSQLDump**(*connection*, *query*, *file*, *character_data=True*)
This class is the MS SQL Server implementation of the BCP Dump operation.

Parameters

- **connection** (Connection) – the Connection object that points to the database from which we want to export data
- **query** (str) – the query defining the data to be exported
- **file** (DataFile) – the file to which the data will be written

command
This method will build the command that will export data from the supplied table to the supplied file.

Return type str

Returns the command that will be passed into the BCP command line utility

execute()
This will run the instance's command via the BCP utility

class bcp.dialects.mssql.**MSSQLLoad**(*connection*, *file*, *table*, *batch_size=10000*, *character_data=True*)
This class is the MS SQL Server implementation of the BCP Load operation.

Parameters

- **connection** (Connection) – the (mssql) Connection object that points to the database from which we want to export data
- **file** (DataFile) – the file whose data should be imported into the target database

- **table** (str) – the into which the data will be written
- **batch_size** (int) – the number of records to read in one commit, defaulted to 10,000
- **character_data** (bool) – allows BCP to use character data, defaulted to True

command
This method will build the command that will import data from the supplied file to the supplied table.

Return type str

Returns the command that will be passed into the BCP command line utility

config
This method will generate a configuration string. It supports the delimiter and batch size options.

Return type str

Returns a BCP formatted configuration string

error
This method will generate an error file string that will write the error file to the BCP_DATA_DIR directory.

Return type str

Returns a BCP formatted error file string

execute()
This will run the instance's command via the BCP utility

CHAPTER 4

Configuration

4.1 Config

This module creates directories (or returns existing directories) to store logs, data and other artifacts.

Note: This application defaults to creating a 'bcp' directory inside of the user's home directory. This would be the value of %USERPROFILE% on Windows or %HOME% on linux. This can be overridden by setting a value for the environmental variable BCP_ROOT_DIR. The structure within this root directory will always be the same.

4.2 Exceptions

This module contains exceptions for this application.

exception bcp.exceptions.**DriverNotSupportedException**
This exception occurs when an unsupported driver is provided to a Connection or BCP object

exception bcp.exceptions.**InvalidCredentialException**
This exception occurs when a username is provided without a password, or vice versa, for an Auth or BCP object

Python Module Index

b

Index