# Contents

1 About 3
2 Installation 5
3 Example usage 7
   3.1 Scripting 7
   3.2 Command line tool 8
4 Contributing 13
5 Functions and classes 15
6 Coding style 27
7 Tips and tricks 29
Index 31
CHAPTER 1

About

CAN BUS tools in Python 3.

- **DBC, KCD, SYM, ARXML 4 and CDD file parsing.**
- **CAN message encoding and decoding.**
- **Simple and extended signal multiplexing.**
- **Diagnostic DID encoding and decoding.**
- **candump output decoder.**
- **Node tester.**
- **C source code generator.**
- **CAN bus monitor.**

Python 2 support is deprecated as Python 3 has better unicode support.

**Project homepage:** https://github.com/eerimoq/cantools

**Documentation:** http://cantools.readthedocs.org/en/latest
CHAPTER 2

Installation

```
pip install cantools
```
CHAPTER 3

Example usage

3.1 Scripting

The example starts by parsing a small DBC-file and printing its messages and signals.

```python
>>> import cantools
>>> from pprint import pprint
>>> db = cantools.database.load_file('tests/files/dbc/motohawk.dbc')
>>> db.messages
[messag('ExampleMessage', 0x1f0, False, 8, 'Example message used as template in MotoHawk models.')]
>>> example_message = db.get_message_by_name('ExampleMessage')
>>> pprint(example_message.signals)
[signal('Enable', 7, 1, 'big_endian', False, 1.0, 0, 0.0, 0.0, '-', False, None, {0: 'Disabled', 1: 'Enabled'}, None), signal('AverageRadius', 6, 6, 'big_endian', False, 0.1, 0, 0.0, 5.0, 'm', False, None, None, None), signal('Temperature', 0, 12, 'big_endian', True, 0.01, 250, 229.53, 270.47, 'degK', False, None, None, None)]
```

The example continues encoding a message and sending it on a CAN bus using the python-can package.

```python
>>> import can
>>> can_bus = can.interface.Bus('vcan0', bustype='socketcan')
>>> data = example_message.encode({'Temperature': 250.1, 'AverageRadius': 3.2, 'Enable': 1})
>>> message = can.Message(arbitration_id=example_message.frame_id, data=data)
>>> can_bus.send(message)
```

Alternatively, a message can be encoded using the `encode_message()` method on the database object.

The last part of the example receives and decodes a CAN message.
>>> message = can_bus.recv()
>>> db.decode_message(message.arbitration_id, message.data)
{'AverageRadius': 3.2, 'Enable': 'Enabled', 'Temperature': 250.09}

See examples for additional examples.

## 3.2 Command line tool

### 3.2.1 The decode subcommand

Decode CAN frames captured with the Linux program candump.

```bash
$ candump vcan0 | cantools decode tests/files dbc/motohawk.dbc
```

ExampleMessage(
  Enable: 'Enabled' - ,
  AverageRadius: 0.0 m ,
  Temperature: 255.92 degK
)

Alternatively, the decoded message can be printed on a single line:

```bash
$ candump vcan0 | cantools decode --single-line tests/files dbc/motohawk.dbc
```

ExampleMessage(Enable: 'Enabled' - ,
  AverageRadius: 0.0 m ,
  Temperature: 255.92 degK)

### 3.2.2 The dump subcommand

Dump given database in a human readable format:

```bash
$ cantools dump tests/files dbc/motohawk.dbc
```

(continues on next page)
Length: 8 bytes
Cycle time: - ms
Senders: PCM1

Layout:

```
Bit

   7  6  5  4  3  2  1  0
+---+---+---+---+---+---+---+
0 |<x|-----------------------x|<--|
+---+---+---+---+---+---+---+
    |  +-- AverageRadius
    +-- Enable

1 |--------------------------|
+---+---+---+---+---+---+---+
2 |--------------x| | | | | |
B +---+---+---+---+---+---+---+
y +-- Temperature
t

3 | | | | | | | |
+---+---+---+---+---+---+---+
4 | | | | | | | |
+---+---+---+---+---+---+---+
5 | | | | | | | |
+---+---+---+---+---+---+---+
6 | | | | | | | |
+---+---+---+---+---+---+---+
7 | | | | | | | |
+---+---+---+---+---+---+---+
```

Signal tree:

```
-- {root}
  +-- Enable
  |   +-- AverageRadius
  |   +-- Temperature

Signal choices:

Enable
    0 Disabled
    1 Enabled
```

3.2.3 The generate C source subcommand

Generate C source code from given database.

The generated code contains:

- Message structs.
- Message pack and unpack functions.
- Signal encode and decode functions.
• Frame id, length, type, cycle time and signal choices defines.

Known limitations:
• The maximum signal size is 64 bits, which in practice is never exceeded.

Below is an example of how to generate C source code from a database. The database is tests/files/dbc/motohawk.dbc.

```
$ cantools generate_c_source tests/files/dbc/motohawk.dbc
Successfully generated motohawk.h and motohawk.c.
```

See motohawk.h and motohawk.c for the contents of the generated files.

In the next example we use --database-name to set a custom namespace for all generated types, defines and functions. The output file names are also changed by this option.

```
$ cantools generate_c_source --database-name my_database_name tests/files/dbc/
˓→motohawk.dbc
Successfully generated my_database_name.h and my_database_name.c.
```

See my_database_name.h and my_database_name.c for the contents of the generated files.

In the last example we use --no-floating-point-numbers to generate code without floating point types, i.e. float and double.

```
$ cantools generate_c_source --no-floating-point-numbers tests/files/dbc/motohawk.dbc
Successfully generated motohawk.h and motohawk.c.
```

See motohawk_no_floating_point_numbers.h and motohawk_no_floating_point_numbers.c for the contents of the generated files.

Other C code generators:
• http://www(coderdbc.com
• https://github.com/howerj/dbcc
• https://github.com/lonkamikaze/hsk-libs/blob/master/scripts/dbc2c.awk
• https://sourceforge.net/projects/comframe/

### 3.2.4 The monitor subcommand

Monitor CAN bus traffic in a text based user interface.

```
$ cantools monitor tests/files/dbc/motohawk.dbc
```
The menu at the bottom of the monitor shows the available commands.

- Quit: Quit the monitor. Ctrl-C can be used as well.
- Filter: Only display messages matching given regular expression. Press <Enter> to return to the menu from the filter input line.
- Play/Pause: Toggle between playing and paused (or running and freeze).
- Reset: Reset the monitor to its initial state.
1. Fork the repository.

2. Install prerequisites.

   ```bash
   pip install -r requirements.txt
   ```

3. Implement the new feature or bug fix.

4. Implement test case(s) to ensure that future changes do not break legacy.

5. Run the tests.

   ```bash
   make test
   ```

6. Create a pull request.
Functions and classes

cantools.database.load_file(filename, database_format=None, encoding=None, frame_id_mask=None, strict=True, cache_dir=None)

Open, read and parse given database file and return a can.Database or diagnostics.Database object with its contents.

database_format is one of 'arxml', 'dbc', 'kcd', 'sym', 'cdd' and None. If None, the database format is selected based on the filename extension as in the table below.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Database format</th>
</tr>
</thead>
<tbody>
<tr>
<td>.arxml</td>
<td>'arxml'</td>
</tr>
<tr>
<td>.dbc</td>
<td>'dbc'</td>
</tr>
<tr>
<td>.kcd</td>
<td>'kcd'</td>
</tr>
<tr>
<td>.sym</td>
<td>'sym'</td>
</tr>
<tr>
<td>.cdd</td>
<td>'cdd'</td>
</tr>
<tr>
<td>&lt;unknown&gt;</td>
<td>None</td>
</tr>
</tbody>
</table>

encoding specifies the file encoding. If None, the encoding is selected based on the database format as in the table below. Use open() and load() if platform dependent encoding is desired.

<table>
<thead>
<tr>
<th>Database format</th>
<th>Default encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>'arxml'</td>
<td>'utf-8'</td>
</tr>
<tr>
<td>'dbc'</td>
<td>'cp1252'</td>
</tr>
<tr>
<td>'kcd'</td>
<td>'utf-8'</td>
</tr>
<tr>
<td>'sym'</td>
<td>'cp1252'</td>
</tr>
<tr>
<td>'cdd'</td>
<td>'utf-8'</td>
</tr>
<tr>
<td>None</td>
<td>'utf-8'</td>
</tr>
</tbody>
</table>

cache_dir specifies the database cache location in the file system. Give as None to disable the cache. By default the cache is disabled. The cache key is the contents of given file. Using a cache will significantly reduce the load time when reloading the same file. The cache directory is automatically created if it does not exist. Remove the cache directory cache_dir to clear the cache.
See `load_string()` for descriptions of other arguments.

Raises an `UnsupportedDatabaseFormatError` exception if given file does not contain a supported database format.

```python
>>> db = cantools.database.load_file('foo.dbc')
>>> db.version
'1.0'
```

cantools.database.dump_file(database, filename, database_format=None, encoding=None)

Dump given database *database* to given file *filename*.

See `load_file()` for descriptions of other arguments.

```python
>>> db = cantools.database.load_file('foo.dbc')
>>> cantools.database.dump_file(db, 'bar.dbc')
```

cantools.database.load_string(string, database_format=None, frame_id_mask=None, strict=True)

Parse given database string and return a `can.Database` or `diagnostics.Database` object with its contents.

`database_format` may be one of 'arxml', 'dbc', 'kcd', 'sym', 'cdd' or None, where None means transparent format.

See `can.Database` for a description of `strict`.

Raises an `UnsupportedDatabaseFormatError` exception if given string does not contain a supported database format.

```python
>>> with open('foo.dbc') as fin:
...     db = cantools.database.load_string(fin.read())
>>> db.version
'1.0'
```

cantools.database.load(fp, database_format=None, frame_id_mask=None, strict=True)

Read and parse given database file-like object and return a `can.Database` or `diagnostics.Database` object with its contents.

See `load_string()` for descriptions of other arguments.

Raises an `UnsupportedDatabaseFormatError` exception if given file-like object does not contain a supported database format.

```python
>>> with open('foo.kcd') as fin:
...     db = cantools.database.load(fin)
>>> db.version
None
```

class cantools.database.can.Database(messages=None, nodes=None, buses=None, version=None, dbc_specifics=None, frame_id_mask=None, strict=True)

This class contains all messages, signals and definitions of a CAN network.

The factory functions `load()`, `load_file()` and `load_string()` returns instances of this class.

If `strict` is True an exception is raised if any signals are overlapping or if they don’t fit in their message.

**messages**

A list of messages in the database.
Use \texttt{get_message_by_frame_id()} or \texttt{get_message_by_name()} to find a message by its frame id or name.

\textbf{nodes}
\begin{itemize}
  \item A list of nodes in the database.
\end{itemize}

\textbf{buses}
\begin{itemize}
  \item A list of CAN buses in the database.
\end{itemize}

\textbf{version}
\begin{itemize}
  \item The database version, or \texttt{None} if unavailable.
\end{itemize}

\textbf{dbc}
\begin{itemize}
  \item An object containing dbc specific properties like e.g. attributes.
\end{itemize}

\textbf{add_arxml} \((fp)\)
\begin{itemize}
  \item Read and parse ARXML data from given file-like object and add the parsed data to the database.
\end{itemize}

\textbf{add_arxml_file} \((filename, encoding='utf-8')\)
\begin{itemize}
  \item Open, read and parse ARXML data from given file and add the parsed data to the database.
  \item \textit{encoding} specifies the file encoding.
\end{itemize}

\textbf{add_arxml_string} \((string)\)
\begin{itemize}
  \item Parse given ARXML data string and add the parsed data to the database.
\end{itemize}

\textbf{add dbc} \((fp)\)
\begin{itemize}
  \item Read and parse DBC data from given file-like object and add the parsed data to the database.
\end{itemize}

\begin{verbatim}
>>> db = cantools.database.Database()
>>> with open ('foo.dbc', 'r') as fin:
...     db.add_dbc(fin)
\end{verbatim}

\textbf{add dbc file} \((filename, encoding='cp1252')\)
\begin{itemize}
  \item Open, read and parse DBC data from given file and add the parsed data to the database.
  \item \textit{encoding} specifies the file encoding.
\end{itemize}

\begin{verbatim}
>>> db = cantools.database.Database()
>>> db.add_dbc_file('foo.dbc')
\end{verbatim}

\textbf{add dbc string} \((string)\)
\begin{itemize}
  \item Parse given DBC data string and add the parsed data to the database.
\end{itemize}

\begin{verbatim}
>>> db = cantools.database.Database()
>>> with open ('foo.dbc', 'r') as fin:
...     db.add_dbc_string(fin.read())
\end{verbatim}

\textbf{add_kcd} \((fp)\)
\begin{itemize}
  \item Read and parse KCD data from given file-like object and add the parsed data to the database.
\end{itemize}

\textbf{add_kcd_file} \((filename, encoding='utf-8')\)
\begin{itemize}
  \item Open, read and parse KCD data from given file and add the parsed data to the database.
  \item \textit{encoding} specifies the file encoding.
\end{itemize}

\textbf{add_kcd_string} \((string)\)
\begin{itemize}
  \item Parse given KCD data string and add the parsed data to the database.
\end{itemize}

\textbf{add_sym} \((fp)\)
\begin{itemize}
  \item Read and parse SYM data from given file-like object and add the parsed data to the database.
\end{itemize}
add_sym_file(filename, encoding='utf-8')
Open, read and parse SYM data from given file and add the parsed data to the database.

encoding specifies the file encoding.

add_sym_string(string)
Parse given SYM data string and add the parsed data to the database.

as_dbc_string()
Return the database as a string formatted as a DBC file.

as_kcd_string()
Return the database as a string formatted as a KCD file.

get_message_by_name(name)
Find the message object for given name name.

get_message_by_frame_id(frame_id)
Find the message object for given frame id frame_id.

get_node_by_name(name)
Find the node object for given name name.

get_bus_by_name(name)
Find the bus object for given name name.

encode_message(frame_id_or_name, data, scaling=True, padding=False, strict=True)
Encode given signal data data as a message of given frame id or name frame_id_or_name. data is a dictionary of signal name-value entries.

If scaling is False no scaling of signals is performed.

If padding is True unused bits are encoded as 1.

If strict is True all signal values must be within their allowed ranges, or an exception is raised.

>>> db.encode_message(158, {'Bar': 1, 'Fum': 5.0})
'b\x01\x45\x23\x00\x11'

decode_message(frame_id_or_name, data, decode_choices=True, scaling=True)
Decode given signal data data as a message of given frame id or name frame_id_or_name. Returns a dictionary of signal name-value entries.

If decode_choices is False scaled values are not converted to choice strings (if available).

If scaling is False no scaling of signals is performed.

>>> db.decode_message(158, b'\x01\x45\x23\x00\x11')
{'Bar': 1, 'Fum': 5.0}

refresh()
Refresh the internal database state.

This method must be called after modifying any message in the database to refresh the internal lookup tables used when encoding and decoding messages.
A CAN message with frame id, comment, signals and other information.

If `strict` is `True` an exception is raised if any signals are overlapping or if they don’t fit in the message.

- **frame_id**
  The message frame id.

- **is_extended_frame**
  True if the message is an extended frame, `False` otherwise.

- **name**
  The message name as a string.

- **length**
  The message data length in bytes.

- **signals**
  A list of all signals in the message.

- **comment**
  The message comment, or `None` if unavailable.

- **senders**
  A list of all sender nodes of this message.

- **send_type**
  The message send type, or `None` if unavailable.

- **cycle_time**
  The message cycle time, or `None` if unavailable.

- **dbcs**
  An object containing dbc specific properties like e.g. attributes.

- **bus_name**
  The message bus name, or `None` if unavailable.

- **protocol**
  The message protocol, or `None` if unavailable. Only one protocol is currently supported; 'j1939'.

- **signal_tree**
  All signal names and multiplexer ids as a tree. Multiplexer signals are dictionaries, while other signals are strings.

```python
>>> foo = db.get_message_by_name('Foo')
>>> foo.signal_tree
['Bar', 'Fum']
>>> bar = db.get_message_by_name('Bar')
>>> bar.signal_tree
[{'A': {0: ['C', 'D'], 1: ['E']}}, 'B']
```

- **signal_tree_string()**
  Returns the message signal tree as a string.

- **signal_choices_string()**
  Returns the signal choices as a string.
layout_string\((signal\_names=True)\)
Returns the message layout as an ASCII art string. Each signal is an arrow from LSB x to MSB <. Overlapping signal bits are set to X.

Set signal\_names to False to hide signal names.

```
+---+---+---+---+---+---+---+---+
0  | | | | | |<----------|
+---+---+---+---+---+---+---+---+
1  |------x| | | | |<-x| |
+---+---+---+---+---+---+---+---+
   | +-- Bar
   +-- Foo
+---+---+---+---+---+---+---+---+
2  | | | | | | | |
+---+---+---+---+---+---+---+---+
B y t e 3  |----XXXXXXX---x| | | | |
+---+---+---+---+---+---+---+---+
   +-- Fie
+---+---+---+---+---+---+---+---+
4  | | | | | | | |
+---+---+---+---+---+---+---+---+
5  | | |<----------------------|
+---+---+---+---+---+---+---+---+
   +-- Fum
+---+---+---+---+---+---+---+---+
6  | | | | | | | |
+---+---+---+---+---+---+---+---+
7  | | | | | | | |
+---+---+---+---+---+---+---+---+
```

encode\((data, scaling=True, padding=False, strict=True)\)
Encode given data as a message of this type.

If scaling is False no scaling of signals is performed.

If padding is True unused bits are encoded as 1.

If strict is True all signal values must be within their allowed ranges, or an exception is raised.

```
>>> foo = db.get_message_by_name('Foo')
>>> foo.encode({'Bar': 1, 'Fum': 5.0})
b'\x01\x45\x23\x00\x11'
```

decode\((data, decode\_choices=True, scaling=True)\)
Decode given data as a message of this type.

If decode\_choices is False scaled values are not converted to choice strings (if available).

If scaling is False no scaling of signals is performed.

```
>>> foo = db.get_message_by_name('Foo')
>>> foo.decode(b'\x01\x45\x23\x00\x11')
{'Bar': 1, 'Fum': 5.0}
```

is\_multiplexed()
Returns True if the message is multiplexed, otherwise False.
```python
>>> foo = db.get_message_by_name('Foo')
>>> foo.is_multiplexed()
False
>>> bar = db.get_message_by_name('Bar')
>>> bar.is_multiplexed()
True
```

**refresh** *(strict=None)*

Refresh the internal message state.

If `strict` is `True` an exception is raised if any signals are overlapping or if they don’t fit in the message. This argument overrides the value of the same argument passed to the constructor.

```python
class cantools.database.can.Signal(name, start, length, byte_order='little_endian',
is_signed=False, initial=None, scale=1, offset=0, minimum=None, maximum=None, unit=None, choices=None, dbc_specifics=None, comment=None, receivers=None, is_multiplexer=False, multiplexer_ids=None, multiplexer_signal=None, is_float=False, decimal=None)
```

A CAN signal with position, size, unit and other information. A signal is part of a message.

Signal bit numbering in a message:

<table>
<thead>
<tr>
<th>Byte:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit:</td>
<td>7</td>
<td>0</td>
<td>15</td>
<td>8</td>
<td>23</td>
<td>16</td>
<td>31</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Big endian signal with start bit 2 and length 5 (0=LSB, 4=MSB):

<table>
<thead>
<tr>
<th>Byte:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit:</td>
<td>7</td>
<td>0</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

Little endian signal with start bit 2 and length 9 (0=LSB, 8=MSB):

<table>
<thead>
<tr>
<th>Byte:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit:</td>
<td>7</td>
<td>0</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

**name**

The signal name as a string.

**start**

The start bit position of the signal within its message.

**length**

The length of the signal in bits.

**byte_order**

Signal byte order as 'little_endian' or 'big_endian'.
is_signed
   True if the signal is signed, False otherwise. Ignore this attribute if is_float is True.

is_float
   True if the signal is a float, False otherwise.

initial
   The initial value of the signal, or None if unavailable.

scale
   The scale factor of the signal value.

offset
   The offset of the signal value.

minimum
   The minimum value of the signal, or None if unavailable.

maximum
   The maximum value of the signal, or None if unavailable.

decimal
   The high precision values of scale, offset, minimum and maximum.

   See Decimal for more details.

unit
   The unit of the signal as a string, or None if unavailable.

choices
   A dictionary mapping signal values to enumerated choices, or None if unavailable.

dbc
   An object containing dbc specific properties like e.g. attributes.

comment
   The signal comment, or None if unavailable.

receivers
   A list of all receiver nodes of this signal.

is_multiplexer
   True if this is the multiplexer signal in a message, False otherwise.

multiplexer_ids
   The multiplexer ids list if the signal is part of a multiplexed message, None otherwise.

multiplexer_signal
   The multiplexer signal if the signal is part of a multiplexed message, None otherwise.

class cantools.database.can.signal.Decimal(scale=None, offset=None, minimum=None, maximum=None)

   Holds the same values as scale, offset, minimum and maximum, but as decimal.Decimal instead of int and float for higher precision (no rounding errors).

scale
   The scale factor of the signal value as decimal.Decimal.

offset
   The offset of the signal value as decimal.Decimal.

minimum
   The minimum value of the signal as decimal.Decimal, or None if unavailable.
maximum
The maximum value of the signal as `decimal.Decimal`, or `None` if unavailable.

class cantools.database.diagnostics.Database(dids=None)
This class contains all DIDs.
The factory functions `load()`, `load_file()` and `load_string()` returns instances of this class.

dids
A list of DIDs in the database.

add_cdd(fp)
Read and parse CDD data from given file-like object and add the parsed data to the database.

add_cdd_file(filename, encoding='utf-8')
Open, read and parse CDD data from given file and add the parsed data to the database.

encoding specifies the file encoding.

add_cdd_string(string)
Parse given CDD data string and add the parsed data to the database.

get_did_by_name(name)
Find the DID object for given name `name`.

get_did_by_identifier(identifier)
Find the DID object for given identifier `identifier`.

refresh()
Refresh the internal database state.

This method must be called after modifying any DIDs in the database to refresh the internal lookup tables used when encoding and decoding DIDs.

class cantools.database.diagnostics.Did(identifier, name, length, datas)
A DID with identifier and other information.

identifier
The did identifier as an integer.

name
The did name as a string.

length
The did name as a string.

datas
The did datas as a string.

encode(data, scaling=True)
Encode given data as a DID of this type.

If `scaling` is `False` no scaling of datas is performed.

```python
>>> foo = db.get_did_by_name('Foo')
>>> foo.encode({'Bar': 1, 'Fum': 5.0})
b\x01\x45\x23\x00\x11
```

decode(data, decode_choices=True, scaling=True)
Decode given data as a DID of this type.

If `decode_choices` is `False` scaled values are not converted to choice strings (if available).

If `scaling` is `False` no scaling of datas is performed.
>>> foo = db.get_did_by_name('Foo')
>>> foo.decode(b'(\x01\x45\x23\x00\x11')
{'Bar': 1, 'Fum': 5.0}

refresh()
Refresh the internal DID state.

```python
class cantools.database.diagnostics.Data(name, start, length, byte_order='little_endian',
                                      scale=1, offset=0, minimum=None, maximum=None, unit=None, choices=None)
```

A data data with position, size, unit and other information. A data is part of a DID.

- **name**
  The data name as a string.

- **start**
  The start bit position of the data within its DID.

- **length**
  The length of the data in bits.

- **byte_order**
  Data byte order as 'little_endian' or 'big_endian'.

- **scale**
  The scale factor of the data value.

- **offset**
  The offset of the data value.

- **minimum**
  The minimum value of the data, or None if unavailable.

- **maximum**
  The maximum value of the data, or None if unavailable.

- **unit**
  The unit of the data as a string, or None if unavailable.

- **choices**
  A dictionary mapping data values to enumerated choices, or None if unavailable.

```python
class cantools.database.UnsupportedDatabaseFormatError(e_arxml, e dbc, e kcd, e_sym, e cdd)
```

This exception is raised when load_file(), load() and load_string() are unable to parse given database file or string.

```python
class cantools.tester.Tester(dut_name, database, can_bus, bus_name=None,
                             on_message=None, decode_choices=True, scaling=True,
                             padding=False)
```

Test given node dut_name on given CAN bus bus_name.

database is a Database instance.

can_bus a CAN bus object, normally created using the python-can package.

The on_message callback is called for every successfully decoded received message. It is called with one argument, an DecodedMessage instance.

Here is an example of how to create a tester:
start()  
Start the tester. Starts sending enabled periodic messages.

```python
>>> tester.start()
```

stop()  
Stop the tester. Periodic messages will not be sent after this call. Call start() to resume a stopped tester.

```python
>>> tester.stop()
```

messages  
Set and get signals in messages. Set signals takes effect immediately for started enabled periodic messages. Call send() for other messages.

```python
>>> periodic_message = tester.messages['PeriodicMessage1']
>>> periodic_message  
{'Signal1': 0, 'Signal2': 0}
>>> periodic_message['Signal1'] = 1
>>> periodic_message.update({'Signal1': 2, 'Signal2': 5})
>>> periodic_message  
{'Signal1': 2, 'Signal2': 5}
```

enable (message_name)  
Enable given message message_name and start sending it if its periodic and the tester is running.

```python
>>> tester.enable('PeriodicMessage1')
```

disable (message_name)  
Disable given message message_name and stop sending it if its periodic, enabled and the tester is running.

```python
>>> tester.disable('PeriodicMessage1')
```

send (message_name, signals=None)  
Send given message message_name and optional signals signals.

```python
>>> tester.send('Message1', {'Signal2': 10})
>>> tester.send('Message1')
```

expect (message_name, signals=None, timeout=None, discard_other_messages=True)  
Expect given message message_name and signal values signals within timeout seconds.  
Give signals as None to expect any signal values.  
Give timeout as None to wait forever.  
Messages are read from the input queue, and those not matching given message_name and signals are discarded if discard_other_messages is True. flush_input() may be called to discard all old messages in the input queue before calling the expect function.
Returns the expected message, or `None` on timeout.

```python
>>> tester.expect('Message2', {'Signal1': 13})
{'Signal1': 13, 'Signal2': 9}
```

`flush_input()`
Flush, or discard, all messages in the input queue.

class cantools.tester.DecodedMessage(name, signals)
A decoded message.

- **name**
  - Message name.

- **signals**
  - Message signals.
The coding style for this package is defined as below. The rules are based on my personal preference.

- Blank lines before and after statements (if, while, return, . . .) (1), unless at beginning or end of another statement or file (8).
- Two blank lines between file level definitions (2).
- Space before and after operators (3), except for keyword arguments where no space is allowed (4).
- One import per line (5).
- Comments and doc strings starts with capital letter and ends with a period, that is, just as sentences (6).
- Blank line after doc strings (7).
- Maximum line length of 90 characters, but aim for less than 80.
- All function arguments on one line, or one per line.
- Class names are CamelCase. Underscore is not allowed.
- Function and variable names are lower case with underscore separating words.

```python
import sys
from os import path  # (5)
from os import getcwd # (5)

def foo(bars, fum=None):  # (4)
    """This is a doc string.""" # (6)
    
    fies = []  # (7)
    kam = path.join(getcwd(), '..') # (3)
    
    for bar in bars:
        if len(bar) == 1):  # (8)
```
```python
fies.append(ham + 2 * bar)  # (3)
# This is a comment.  # (6)
if fum in None:
    fum = 5  # (3)
else:
    fum += 1  # (3)
fies *= fum  # (3)
return fies  # (2)
```

```python
def goo():
    return True
```
Virtual CAN interface setup in Ubuntu:

```bash
sudo modprobe vcan
sudo ip link add dev vcan0 type vcan
sudo ip link set vcan0 mtu 72 # For CAN-FD
sudo ip link set up vcan0
```
add_arxml()  (cantools.database.can.Database method), 17
add_arxml_file()  (cantools.database.can.Database method), 17
add_arxml_string()  (cantools.database.can.Database method), 17
add_cdd()  (cantools.database.diagnostics.Database method), 23
add_cdd_file()  (cantools.database.diagnostics.Database method), 23
add_cdd_string()  (cantools.database.diagnostics.Database method), 23
add_dbc()  (cantools.database.can.Database method), 17
add_dbc_file()  (cantools.database.can.Database method), 17
add_dbc_string()  (cantools.database.can.Database method), 17
add_kcd()  (cantools.database.can.Database method), 17
add_kcd_file()  (cantools.database.can.Database method), 17
add_kcd_string()  (cantools.database.can.Database method), 17
add_sym()  (cantools.database.can.Database method), 17
add_sym_file()  (cantools.database.can.Database method), 17
add_sym_string()  (cantools.database.can.Database method), 17
as_dbc_string()  (cantools.database.can.Database method), 18
as_kcd_string()  (cantools.database.can.Database method), 18
bus_name  (cantools.database.can.Message attribute), 19
buses  (cantools.database.can.Database attribute), 17
byte_order  (cantools.database.can.Signal attribute), 21
byte_order  (cantools.database.diagnostics.Data attribute), 24
choices  (cantools.database.can.Signal attribute), 22
choices  (cantools.database.diagnostics.Data attribute), 24
comment  (cantools.database.can.Message attribute), 19
comment  (cantools.database.can.Signal attribute), 22
cycle_time  (cantools.database.can.Message attribute), 19
Data  (class in cantools.database.diagnostics), 24
Database  (class in cantools.database.can), 16
Database  (class in cantools.database.diagnostics), 23
datas  (cantools.database.diagnostics.Did attribute), 23
dbc  (cantools.database.can.Database attribute), 17
dbc  (cantools.database.can.Message attribute), 19
dbc  (cantools.database.can.Signal attribute), 22
decimal  (cantools.database.can.Signal attribute), 22
Decimal  (class in cantools.database.can.signal), 22
decode()  (cantools.database.can.Message method), 20
decode()  (cantools.database.diagnostics.Did method), 23
decode_message()  (cantools.database.can.Database method), 18
DecodedMessage  (class in cantools.tester), 26
DecodedMessage.name  (in module cantools.tester), 26
DecodedMessage.signals  (in module cantools.tester), 26
Did  (class in cantools.database.diagnostics), 23
dids (cantools.database.diagnostics.Database attribute), 23
disable() (cantools.tester.Tester method), 25
dump_file() (in module cantools.database), 16

E
enable() (cantools.tester.Tester method), 25
cantools.database.can.Message method), 20
cantools.database.diagnostics.Did method), 23
cantools.database.can.Database method), 18
expect() (cantools.tester.Tester method), 25

F
flush_input() (cantools.tester.Tester method), 26
frame_id (cantools.database.can.Message attribute), 19

G
get_bus_by_name() (cantools.database.can.Database method), 18
cantools.database.diagnostics.Database method), 23
cantools.database.diagnostics.Database method), 23
cantools.database.can.Database method), 18
cantools.database.can.Database method), 18
cantools.database.can.Database method), 18

I
identifier (cantools.database.diagnostics.Did attribute), 23
initial (cantools.database.can.Signal attribute), 22
is_extended_frame (cantools.database.can.Message attribute), 19
is_float (cantools.database.can.Signal attribute), 22
is_multiplexed() (cantools.database.can.Message method), 20
is_multiplexer (cantools.database.can.Signal attribute), 22
is_signed (cantools.database.can.Signal attribute), 21

L
layout_string() (cantools.database.can.Message method), 19
length (cantools.database.can.Message attribute), 19
length (cantools.database.can.Message attribute), 19
length (cantools.database.can.Message attribute), 21
length (cantools.database.diagnostics.Data attribute), 24
length (cantools.database.diagnostics.Did attribute), 23
load() (in module cantools.database), 16
load_file() (in module cantools.database), 15
load_string() (in module cantools.database), 16

M
maximum (cantools.database.can.Signal attribute), 22
maximum (cantools.database.can.signal.Decimal attribute), 22
maximum (cantools.database.diagnostics.Data attribute), 24
messages (cantools.database.can.Database attribute), 16
messages (cantools.tester.Tester attribute), 25
minimum (cantools.database.can.Signal attribute), 22
minimum (cantools.database.diagnostics.Data attribute), 22
minimum (cantools.database.diagnostics.Data attribute), 24
multiplexer_ids (cantools.database.can.Signal attribute), 22
multiplexer_signal (cantools.database.can.Signal attribute), 22

N
name (cantools.database.can.Message attribute), 19
name (cantools.database.can.Signal attribute), 21
name (cantools.database.diagnostics.Data attribute), 24
name (cantools.database.diagnostics.Did attribute), 23
nodes (cantools.database.can.Database attribute), 17

O
offset (cantools.database.can.Signal attribute), 22
offset (cantools.database.can.signal.Decimal attribute), 22
offset (cantools.database.diagnostics.Data attribute), 24

P
protocol (cantools.database.can.Message attribute), 19

R
receivers (cantools.database.can.Signal attribute), 22
refresh() (cantools.database.can.Database method), 18
refresh() (cantools.database.can.Message method), 21
refresh() (cantools.database.diagnostics.Database method), 23
refresh() (cantools.database.diagnostics.Did method), 24

S
scale (cantools.database.can.Signal attribute), 22
scale (cantools.database.can.signal.Decimal attribute), 22
scale (cantools.database.diagnostics.Data attribute), 24
send() (cantools.tester.Tester method), 25
send_type (cantools.database.can.Message attribute), 19
senders (cantools.database.can.Message attribute), 19
Signal (class in cantools.database.can), 21
signal_choices_string() (cantools.database.can.Message method), 19
signal_tree (cantools.database.can.Message attribute), 19
signal_tree_string() (cantools.database.can.Message method), 19
signals (cantools.database.can.Message attribute), 19
start (cantools.database.can.Signal attribute), 21
start (cantools.database.diagnostics.Data attribute), 24
start() (cantools.tester.Tester method), 25
stop() (cantools.tester.Tester method), 25

T
Tester (class in cantools.tester), 24

U
unit (cantools.database.can.Signal attribute), 22
unit (cantools.database.diagnostics.Data attribute), 24
UnsupportedDatabaseFormatError (class in cantools.database), 24

V
version (cantools.database.can.Database attribute), 17