Unittest for `stringtools`

August 15, 2025

Contents

1 Test Information

1.1 Test Candidate Information

The Module stringtools is designed to support functionality for strings (e.g. transfer strings via a bytestream, compressing, extracting, ...). For more Information read the sphinx documentation.

Library Information	
Name	stringtools
State	Released
Supported Interpreters	python3
Version	40bca0906a91f424dc2e76abb22d5065
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	68554cbff580852aac9c6e233a23a458
Testruns with	python 3.13.5 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	Debian GNU/Linux 13 trixie
Hostname	ahorn
Kernel	6.12.38+deb13-amd64 (#1 SMP PREEMPT_DYNAMIC Debian 6.12.38-1 (2025-07-16))
Machine	x86_64
Path	/home/dirk/work/unittest_collection/stringtools
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun with python 3.13.5 (final)

Number of tests	**21**
Number of successfull tests	**21**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	0.019s

2.2 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
stringtools	100.0%	96.9%
stringtools.__init__.py	100.0%	
stringtools.csp.py	100.0%	
stringtools.stp.py	100.0%	

3 Tested Requirements

3.1 Stream Definition

3.1.1 Physical representation

Description

The library stringtools shall have a method physical_repr, transforming a float or integer value to a string with a 1 to 3 digit value followed by the physical prefix for the unit.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.1!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,191
Finished-Time:	2025-08-15 21:12:59,194
Time-Consumption	0.003s

Testsummary:	
Success	Physical representation for 1.17e-10 is correct (Content '117p' and Type is <class 'str'>).
Success	Physical representation for 5.4e-08 is correct (Content '54n' and Type is <class 'str'>).
Success	Physical representation for 2.53e-05 is correct (Content '25.3u' and Type is <class 'str'>).
Success	Physical representation for 0.1 is correct (Content '100m' and Type is <class 'str'>).
Success	Physical representation for 0 is correct (Content '0' and Type is <class 'str'>).
Success	Physical representation for 1 is correct (Content '1' and Type is <class 'str'>).
Success	Physical representation for 1000 is correct (Content '1k' and Type is <class 'str'>).
Success	Physical representation for 1005001 is correct (Content '1.01M' and Type is <class 'str'>).
Success	Physical representation for 1004000000 is correct (Content '1G' and Type is <class 'str'>).
Success	Physical representation for 1003000000000 is correct (Content '1T' and Type is <class 'str'>).
Success	Physical representation for 10000000000000000 is correct (Content '10P' and Type is <class 'str'>).
Success	Physical representation for 17.17 is correct (Content '17.17' and Type is <class 'str'>).
Success	Physical representation for 117000 is correct (Content '117k' and Type is <class 'str'>).
Success	Physical representation for 117.17 is correct (Content '117.2' and Type is <class 'str'>).

3.1.2 Time representation

Description

The library stringtools shall have a method physical_repr, transforming an integer value to a time string like HH:MM:SS.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.2!

Testrun:	python 3.13.5 (final)

Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,195
Finished-Time:	2025-08-15 21:12:59,196
Time-Consumption	0.001s

Testsummary:	
Success	Time representation for 59 is correct (Content '00:59' and Type is <class 'str'>).
Success	Time representation for 60 is correct (Content '01:00' and Type is <class 'str'>).
Success	Time representation for 3599 is correct (Content '59:59' and Type is <class 'str'>).
Success	Time representation for 3600 is correct (Content '01:00:00' and Type is <class 'str'>).
Success	Time representation for 86399 is correct (Content '23:59:59' and Type is <class 'str'>).
Success	Time representation for 86400 is correct (Content '1D' and Type is <class 'str'>).
Success	Time representation for 86459 is correct (Content '1D 00:59' and Type is <class 'str'>).
Success	Time representation for 90000 is correct (Content '1D 01:00:00' and Type is <class 'str'>).

3.1.3 Fraction representation

Description

The library `stringtools` shall have a method `frac_repr`, transforming a float or integer value to a fraction string with a limited denominator.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.3!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,196
Finished-Time:	2025-08-15 21:12:59,197
Time-Consumption	0.001s

Testsummary:	
Success	Fraction representation for 17.4 is correct (Content '87/5' and Type is <class 'str'>).
Success	Fraction representation for 0.25 is correct (Content '1/4' and Type is <class 'str'>).
Success	Fraction representation for 0.1 is correct (Content '1/10' and Type is <class 'str'>).
Success	Fraction representation for 0.01666667 is correct (Content '1/60' and Type is <class 'str'>).

3.2 Human readable value representations

3.3 Stream to Human readable String

3.3.1 Hexadecimal Values

Description

A Stream shall be converted to a human readable String containing all bytes as hexadecimal values seperated by a Space.

Reason for the implementation
Make non printable characters printable.

Fitcriterion
A stream shall be converted at least once and the hex values shall exist in the returnvalue in the correct order.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.4!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,197
Finished-Time:	2025-08-15 21:12:59,198
Time-Consumption	0.001s

Testsummary:

Info	Checking test pattern de ad be ef (<class 'bytes'>).
Success	Pattern included all relevant information in the correct order.

3.3.2 Number of Bytes

Description
The Length of a Stream surrounded by brakets shall be included in the human readable string.

Reason for the implementation
Show the length of a Stream without counting the seperated values.

Fitcriterion
The described pattern including the decimal number of bytes is included in the string for at least one Stream.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.5!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,198
Finished-Time:	2025-08-15 21:12:59,198
Time-Consumption	0.000s

Testsummary:

Info	Checking test pattern with length 4.
Success	'(4)' is in '(4): de ad be ef' at position 0

3.3.3 CRLF-Filter

Description
The module stringtools shall have a method to replace carriage return and line feed to their escaped representation.

Reason for the implementation
Replace these characters to make output printable (e.g. for logging a string based protocol).

Fitcriterion
Filter at least one string and check at least one CR and one LF representation.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.6!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,198
Finished-Time:	2025-08-15 21:12:59,199
Time-Consumption	0.000s

Testsummary:

Info	Checking test pattern with length 4.
Success	Returnvalue of linefeed_filter is correct (Content b'test//r//n123//r//n' and Type is <class 'bytes'>).

3.4 Stream Compression

3.4.1 Compress

Description
The module stringtools shall have a method compressing a Stream with gzip.

Reason for the implementation
Speed up transfer with low transfer rate.

Fitcriterion
Compressed Stream is extractable and results in the original data.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.7!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,199

Finished-Time:	2025-08-15 21:12:59,200
Time-Consumption	0.001s

Testsummary:	
Info	Compressing Streams result in differnt streams with the same input stream. Therefore the test will compare the decompressed data.
Info	Compressing stream: (30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff
Info	Extracting stream: (26): 1f 8b 08 00 bb 86 9f 68 02 ff 63 60 40 01 ff 51 01 00 2d 8a 7d de 1e 00 00 00
Success	Extracted data is correct (Content (30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff and Type is <class 'bytes'>).

3.4.2 Extract

Description

The module stringtools shall have a method extracting a Stream with gzip.

Reason for the implementation

Speed up transfer with low transfer rate.

Fitcriterion

Extracted Stream is equal to the original compressed data.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.8!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,200
Finished-Time:	2025-08-15 21:12:59,200
Time-Consumption	0.000s

Testsummary:	
Info	Extracting stream: (26): 1f 8b 08 00 34 e0 04 5d 02 ff 63 60 40 01 ff 51 01 00 2d 8a 7d de 1e 00 00 00
Success	Extracted data is correct (Content '(30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff' and Type is <class 'str'>).

3.5 Carriagereturn Seperation Protocol (CSP)

3.5.1 Frame creation

Description

The CSP module shall support a method to create a Frame from a stream.

Reason for the implementation

Simple message or frame generation for streams (e.g. Keyboard (user input), RFID-Reader, . . .).

Fitcriterion

Creation of a testframe and checking the result.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.9!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,200
Finished-Time:	2025-08-15 21:12:59,201
Time-Consumption	0.000s
Testsummary:	
Info	Creating testframe for 'b':testframe: for csp"
Success	CSP-Frame is correct (Content b':testframe: for csp/n' and Type is <class 'bytes'>).

3.5.2 Frame creation error

Description

The Frame creation Method shall raise ValueError, if a frame separation character is in the Source-String.

Reason for the implementation

String including separation charcter will be splitted in pieces while processing after transport.

Fitcriterion

ValueErroro is raised for at least one String including the separation character.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.10!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,201
Finished-Time:	2025-08-15 21:12:59,201
Time-Consumption	0.000s
Testsummary:	
Info	Creating testframe for 'b':testframe: for csp"
Success	CSP-Frame is correct (Content <class 'ValueError'> and Type is <class 'type'>).

3.5.3 Frame processing

Description

The CSP Module shall support a class including a method to process stream snipets of variable length. This Method shall return an empty list, if no frame has been detected, otherwise it shall return a list including detected frame(s).

Reason for the implementation

Support message analysis of a stream with every size.

Fitcriterion

At least one frame given in at least two snippets is identified correctly.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.11!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,201
Finished-Time:	2025-08-15 21:12:59,202
Time-Consumption	0.001s

Testsummary:

Info	Processing testframe: 'b':testframe: for csp/n" in two snippets
Success	First processed CSP-Snippet is correct (Content [] and Type is <class 'list'>).
Success	Final processed CSP-Frame is correct (Content [b':testframe: for csp'] and Type is <class 'list'>).

3.5.4 Frame processing - Input data type error

Description

If the input data is not bytes for python3 or str for python 2, the process method shall raise TypeError.

Reason for the implementation

Type restriction.

Fitcriterion

At least the following types return TypeError (list, int, str for python3, unicode for python 2).

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.12!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,202

Finished-Time:	2025-08-15 21:12:59,203
Time-Consumption	0.001s

Testsummary:	
Info	Processing wrong data (list)
Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).
Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).
Info	Processing wrong data (int)
Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).
Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).
Info	Processing wrong data (str)
Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).
Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).

3.6 Serial Transfer Protocol (STP)

3.6.1 Frame creation

Description

A frame creation method shall create a frame out of given input data.

Reason for the implementation

Message or Frame generation for streams (e.g. data transfer via bluetooth, ethernet, ...).

Fitcriterion

Creation of a testframe and checking the result.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.13!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,203
Finished-Time:	2025-08-15 21:12:59,204
Time-Consumption	0.000s

Testsummary:	
Info	Creating testframe for 'b'testframe for stp''
Success	STP-Frame is correct (Content b':<testframe for stp:>' and Type is <class 'bytes'>).

3.6.2 Frame creation - Start pattern and end pattern inside a message

Description

The frame creation method shall support existance of the start or end pattern in the data to be framed.

Reason for the implementation
Possibility to send any kind of data (including the patterns).

Fitcriterion
Creation of a testframe out of data including at least one start pattern and one end pattern and checking the result.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.14!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,204
Finished-Time:	2025-08-15 21:12:59,204
Time-Consumption	0.000s

Testsummary:

Info	Creating testframe including start and end pattern for 'b'testframe for :<stp:>"
Success	STP-Frame is correct (Content b':<testframe for :=<stp:⇒:>' and Type is <class 'bytes'>).

3.6.3 Frame processing

Description
The STP Module shall support a class including a method to process stream snipets of variable length. This Method shall return an empty list, if no frame has been detected, otherwise it shall return a list including detected frame(s).

Reason for the implementation
Support message analysis of a stream with every size.

Fitcriterion
At least one frame given in at least two snippets is identified correctly.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.15!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,204
Finished-Time:	2025-08-15 21:12:59,205
Time-Consumption	0.001s

Testsummary:

Info	Processing testframe: 'b':<testframe for stp:>"
Success	First processed STP snippet is correct (Content [] and Type is <class 'list'>).
Success	Final processed STP snippet is correct (Content [b'testframe for stp'] and Type is <class 'list'>).

3.6.4 Frame processing - Input data type error

Description

If the input data is not bytes for python3 or str for python 2, the process method shall raise TypeError.

Reason for the implementation

Type restriction.

Fitcriterion

At least the following types return TypeError (list, int, str for python3, unicode for python 2).

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.16!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,205
Finished-Time:	2025-08-15 21:12:59,206
Time-Consumption	0.001s

Testsummary:

Info	Processing wrong data (list)
Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).
Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).
Info	Processing wrong data (int)
Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).
Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).
Info	Processing wrong data (str)
Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).
Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).

3.6.5 Frame processing - Start pattern and end pattern inside a message

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.17!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,207
Finished-Time:	2025-08-15 21:12:59,207
Time-Consumption	0.001s

Testsummary:

Info	Processing testframe: 'b':<testframe for :=<stp:⇒:>''
Success	Processed STP-Frame is correct (Content [b'testframe for :<stp:>'] and Type is <class 'list'>).

3.6.6 Frame processing - Data before the start pattern

Description

Data before the start pattern shall be ignored. A warning shall be given to the logger.

Reason for the implementation

Robustness against wrong or corrupted data.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.18!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,207
Finished-Time:	2025-08-15 21:12:59,208
Time-Consumption	0.001s

Testsummary:

Info	Processing testframe: 'b'_:<testframe for stp:>"
Success	Processed STP-Frame is correct (Content [b'testframe for stp'] and Type is <class 'list'>).

3.6.7 Frame processing - Incorrect start patterns

Description

On receiving an incorrect start pattern, STP shall stay in ESCAPE_1, in case of data sync was received twice or back to state IDLE in all other faulty start patterns starting with data sync. A warning shall be given to the logger.

Reason for the implementation

Robustness against wrong or corrupted data.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.19!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,208
Finished-Time:	2025-08-15 21:12:59,209
Time-Consumption	0.001s

Testsummary:

Info	Processing data with an insufficient start pattern.
Success	Return value list if processing incorrect start of frame is correct (Content [[]] and Type is <class 'list'>).
Success	State after processing incorrect start of frame is correct (Content 0 and Type is <class 'int'>).
Info	Processing data with an insufficient start pattern (two times sync).
Success	Return value list if processing data_sync twice is correct (Content [[]] and Type is <class 'list'>).

Success	State after processing data_sync twice is correct (Content 1 and Type is <class 'int'>).

3.6.8 Frame processing - Incorrect end pattern

Description

On receiving an incorrect end pattern, STP shall change to state STORE_DATA, in case of a start pattern, to ESCAPE_1, in case of data sync was received twice or back to state IDLE in all other faulty end patterns starting with data sync. A warning shall be given to the logger.

Reason for the implementation

Robustness against wrong or corrupted data.

Testresult

This test was passed with the state: Success. See also full trace in section A.1.20!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,209
Finished-Time:	2025-08-15 21:12:59,212
Time-Consumption	0.002s

Testsummary:

Info	Processing data with an insufficient end pattern.
Success	Return value list if processing data_sync and data again after start of frame is correct (Content [[]] and Type is <class 'list'>).
Success	State after processing data_sync and data again after start of frame is correct (Content 0 and Type is <class 'int'>).
Success	Buffer size after processing data with insufficient end pattern is correct (Content 0 and Type is <class 'int'>).
Info	Processing data with an insufficient end pattern (start pattern instead of end pattern).
Success	Return value list if processing 2nd start of frame is correct (Content [[]] and Type is <class 'list'>).
Success	State after processing 2nd start of frame is correct (Content 3 and Type is <class 'int'>).
Success	Buffer size after processing 2nd start of frame is correct (Content 0 and Type is <class 'int'>).
Info	Processing data with an insufficient end pattern (two times sync instead of end pattern).
Success	Return value list if processing data_sync twice after start of frame is correct (Content [[]] and Type is <class 'list'>).
Success	State after processing data_sync twice after start of frame is correct (Content 1 and Type is <class 'int'>).

3.6.9 Frame processing - After state corruption

Description

The state of STP shall be set to IDLE, after an unknown state was recognised. The currently processed data shall be processed again. An error shall be given to the logger.

Reason for the implementation

Robustness against wrong or corrupted data.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.21!

Testrun:	python 3.13.5 (final)
Caller:	/home/dirk/work/unittest_collection/stringtools/unittest/src/report/__init__.py (331)
Start-Time:	2025-08-15 21:12:59,212
Finished-Time:	2025-08-15 21:12:59,213
Time-Consumption	0.001s

Testsummary:

Info	Corrupting stp state and processing data.
Success	Return value list if processing start of a frame after state had been corrupted is correct (Content [[]] and Type is <class 'list'>).
Success	State after processing start of a frame after state had been corrupted is correct (Content 3 and Type is <class 'int'>).
Success	Buffer size after corrupting stp state is correct (Content 2 and Type is <class 'int'>).

A Trace for testrun with python 3.13.5 (final)

A.1 Tests with status Info (21)

A.1.1 REQ-0019

Testresult

This test was passed with the state: **Success**.

Success	Physical representation for 1.17e-10 is correct (Content '117p' and Type is <class 'str'>).

```
Result (Physical representation for 1.17e-10): '117p' (<class 'str'>)
Expectation (Physical representation for 1.17e-10): result = '117p' (<class 'str'>)
```

Success	Physical representation for 5.4e-08 is correct (Content '54n' and Type is <class 'str'>).

```
Result (Physical representation for 5.4e-08): '54n' (<class 'str'>)
Expectation (Physical representation for 5.4e-08): result = '54n' (<class 'str'>)
```

Success	Physical representation for 2.53e-05 is correct (Content '25.3u' and Type is <class 'str'>).

```
Result (Physical representation for 2.53e-05): '25.3u' (<class 'str'>)
Expectation (Physical representation for 2.53e-05): result = '25.3u' (<class 'str'>)
```

Success	Physical representation for 0.1 is correct (Content '100m' and Type is <class 'str'>).

```
Result (Physical representation for 0.1): '100m' (<class 'str'>)
Expectation (Physical representation for 0.1): result = '100m' (<class 'str'>)
```

Success	Physical representation for 0 is correct (Content '0' and Type is <class 'str'>).

```
Result (Physical representation for 0): '0' (<class 'str'>)
Expectation (Physical representation for 0): result = '0' (<class 'str'>)
```

Success	Physical representation for 1 is correct (Content '1' and Type is <class 'str'>).

```
Result (Physical representation for 1): '1' (<class 'str'>)
Expectation (Physical representation for 1): result = '1' (<class 'str'>)
```

Success	Physical representation for 1000 is correct (Content '1k' and Type is <class 'str'>).

```
Result (Physical representation for 1000): '1k' (<class 'str'>)
Expectation (Physical representation for 1000): result = '1k' (<class 'str'>)
```

Success	Physical representation for 1005001 is correct (Content '1.01M' and Type is <class 'str'>).

```
Result (Physical representation for 1005001): '1.01M' (<class 'str'>)
```

```
Expectation (Physical representation for 1005001): result = '1.01M' (<class 'str'>)
```

Success Physical representation for 1004000000 is correct (Content '1G' and Type is <class 'str'>).

```
Result (Physical representation for 1004000000): '1G' (<class 'str'>)
Expectation (Physical representation for 1004000000): result = '1G' (<class 'str'>)
```

Success Physical representation for 1003000000000 is correct (Content '1T' and Type is <class 'str'>).

```
Result (Physical representation for 1003000000000): '1T' (<class 'str'>)
Expectation (Physical representation for 1003000000000): result = '1T' (<class 'str'>)
```

Success Physical representation for 10000000000000000 is correct (Content '10P' and Type is <class 'str'>).

```
Result (Physical representation for 10000000000000000): '10P' (<class 'str'>)
Expectation (Physical representation for 10000000000000000): result = '10P' (<class 'str'>)
```

Success Physical representation for 17.17 is correct (Content '17.17' and Type is <class 'str'>).

```
Result (Physical representation for 17.17): '17.17' (<class 'str'>)
Expectation (Physical representation for 17.17): result = '17.17' (<class 'str'>)
```

Success Physical representation for 117000 is correct (Content '117k' and Type is <class 'str'>).

```
Result (Physical representation for 117000): '117k' (<class 'str'>)
Expectation (Physical representation for 117000): result = '117k' (<class 'str'>)
```

Success Physical representation for 117.17 is correct (Content '117.2' and Type is <class 'str'>).

```
Result (Physical representation for 117.17): '117.2' (<class 'str'>)
Expectation (Physical representation for 117.17): result = '117.2' (<class 'str'>)
```

A.1.2 REQ-0020

Testresult

This test was passed with the state: **Success**.

Success Time representation for 59 is correct (Content '00:59' and Type is <class 'str'>).

```
Result (Time representation for 59): '00:59' (<class 'str'>)
Expectation (Time representation for 59): result = '00:59' (<class 'str'>)
```

Success Time representation for 60 is correct (Content '01:00' and Type is <class 'str'>).

```
Result (Time representation for 60): '01:00' (<class 'str'>)
Expectation (Time representation for 60): result = '01:00' (<class 'str'>)
```

Success Time representation for 3599 is correct (Content '59:59' and Type is <class 'str'>).

```
Result (Time representation for 3599): '59:59' (<class 'str'>)
Expectation (Time representation for 3599): result = '59:59' (<class 'str'>)
```

Success Time representation for 3600 is correct (Content '01:00:00' and Type is <class 'str'>).

```
Result (Time representation for 3600): '01:00:00' (<class 'str'>)
Expectation (Time representation for 3600): result = '01:00:00' (<class 'str'>)
```

Success Time representation for 86399 is correct (Content '23:59:59' and Type is <class 'str'>).

```
Result (Time representation for 86399): '23:59:59' (<class 'str'>)
Expectation (Time representation for 86399): result = '23:59:59' (<class 'str'>)
```

Success Time representation for 86400 is correct (Content '1D' and Type is <class 'str'>).

```
Result (Time representation for 86400): '1D' (<class 'str'>)
Expectation (Time representation for 86400): result = '1D' (<class 'str'>)
```

Success Time representation for 86459 is correct (Content '1D 00:59' and Type is <class 'str'>).

```
Result (Time representation for 86459): '1D 00:59' (<class 'str'>)
Expectation (Time representation for 86459): result = '1D 00:59' (<class 'str'>)
```

Success Time representation for 90000 is correct (Content '1D 01:00:00' and Type is <class 'str'>).

```
Result (Time representation for 90000): '1D 01:00:00' (<class 'str'>)
Expectation (Time representation for 90000): result = '1D 01:00:00' (<class 'str'>)
```

A.1.3 REQ-0021

Testresult

This test was passed with the state: **Success**.

Success Fraction representation for 17.4 is correct (Content '87/5' and Type is <class 'str'>).

```
Result (Fraction representation for 17.4): '87/5' (<class 'str'>)
Expectation (Fraction representation for 17.4): result = '87/5' (<class 'str'>)
```

Success Fraction representation for 0.25 is correct (Content '1/4' and Type is <class 'str'>).

```
Result (Fraction representation for 0.25): '1/4' (<class 'str'>)
Expectation (Fraction representation for 0.25): result = '1/4' (<class 'str'>)
```

Success	Fraction representation for 0.1 is correct (Content '1/10' and Type is <class 'str'>).

```
Result (Fraction representation for 0.1): '1/10' (<class 'str'>)
Expectation (Fraction representation for 0.1): result = '1/10' (<class 'str'>)
```

Success	Fraction representation for 0.01666667 is correct (Content '1/60' and Type is <class 'str'>).

```
Result (Fraction representation for 0.01666667): '1/60' (<class 'str'>)
Expectation (Fraction representation for 0.01666667): result = '1/60' (<class 'str'>)
```

A.1.4 REQ-0001

Testresult
This test was passed with the state: **Success**.

Info	Checking test pattern de ad be ef (<class 'bytes'>).

Success	Pattern included all relevant information in the correct order.

```
Return value of hexlify is (4): de ad be ef
Using upper string for comparison: (4): DE AD BE EF
"DE" found in "(4): DE AD BE EF"... Reducing pattern
"AD" found in "AD BE EF"... Reducing pattern
"BE" found in "BE EF"... Reducing pattern
"EF" found in "EF"... Reducing pattern
```

A.1.5 REQ-0002

Testresult
This test was passed with the state: **Success**.

Info	Checking test pattern with length 4.

Success	'(4)' is in '(4): de ad be ef' at position 0

A.1.6 REQ-0005

Testresult
This test was passed with the state: **Success**.

Info	Checking test pattern with length 4.

Success	Returnvalue of linefeed_filter is correct (Content b'test//r//n123//r//n' and Type is <class 'bytes'>).

```
Result (Returnvalue of linefeed_filter): b'test\\r\\n123\\r\\n' (<class 'bytes'>)
```

```
Expectation (Returnvalue of linefeed_filter): result = b'test\\r\\n123\\r\\n' (<class
↪  'bytes'>)
```

A.1.7 REQ-0003

Testresult

This test was passed with the state: **Success**.

Info	Compressing Streams result in differnt streams with the same input stream. Therefore the test will compare the decompressed data.

Info	Compressing stream: (30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

```
GZIP: Finished to compress a string (compression_rate=0.867, consumed_time=0.0s).
```

Info	Extracting stream: (26): 1f 8b 08 00 bb 86 9f 68 02 ff 63 60 40 01 ff 51 01 00 2d 8a 7d de 1e 00 00 00

```
GZIP: Finished to extract a string (compression_rate=0.867, consumed_time=0.0s).
```

Success	Extracted data is correct (Content (30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff and Type is <class 'bytes'>).

```
Result (Extracted data): (30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff
↪  ff ff ff ff ff ff ff ff ff ff (<class 'bytes'>)
```

```
Expectation (Extracted data): result = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
↪  \x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff' (<class 'bytes'>)
```

A.1.8 REQ-0004

Testresult

This test was passed with the state: **Success**.

Info	Extracting stream: (26): 1f 8b 08 00 34 e0 04 5d 02 ff 63 60 40 01 ff 51 01 00 2d 8a 7d de 1e 00 00 00

```
GZIP: Finished to extract a string (compression_rate=0.867, consumed_time=0.0s).
```

Success	Extracted data is correct (Content '(30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff' and Type is <class 'str'>).

```
Result (Extracted data): '(30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff
↪  ff ff ff ff ff ff ff ff ff ff' (<class 'str'>)
```

```
Expectation (Extracted data): result = '(30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff
↪  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff' (<class 'str'>)
```

A.1.9 REQ-0006

Testresult
This test was passed with the state: **Success**.

Info	Creating testframe for 'b':testframe: for csp"

Success	CSP-Frame is correct (Content b':testframe: for csp/n' and Type is <class 'bytes'>).

```
Result (CSP-Frame): b':testframe: for csp\n' (<class 'bytes'>)
Expectation (CSP-Frame): result = b':testframe: for csp\n' (<class 'bytes'>)
```

A.1.10 REQ-0007

Testresult
This test was passed with the state: **Success**.

Info	Creating testframe for 'b':testframe: for csp"

Success	CSP-Frame is correct (Content <class 'ValueError'> and Type is <class 'type'>).

```
Result (CSP-Frame): <class 'ValueError'> (<class 'type'>)
Expectation (CSP-Frame): result = <class 'ValueError'> (<class 'type'>)
```

A.1.11 REQ-0008

Testresult
This test was passed with the state: **Success**.

Info	Processing testframe: 'b':testframe: for csp/n" in two snippets

```
CSP: Leaving data in buffer (to be processed next time): (10): 3a 74 65 73 74 66 72 61 6d 65
CSP: message identified - (19): 3a 74 65 73 74 66 72 61 6d 65 3a 20 66 6f 72 20 63 73 70
```

Success	First processed CSP-Snippet is correct (Content [] and Type is <class 'list'>).

```
Result (First processed CSP-Snippet): [  ] (<class 'list'>)
Expectation (First processed CSP-Snippet): result = [  ] (<class 'list'>)
```

Success	Final processed CSP-Frame is correct (Content [b':testframe: for csp'] and Type is <class 'list'>).

```
Result (Final processed CSP-Frame): [ b':testframe: for csp' ] (<class 'list'>)
Expectation (Final processed CSP-Frame): result = [ b':testframe: for csp' ] (<class 'list'>)
```

A.1.12 REQ-0009

Testresult

This test was passed with the state: **Success**.

Info	Processing wrong data (list)

Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).

```
Result (Wrong data exception): <class 'ValueError'> (<class 'type'>)
Expectation (Wrong data exception): result = <class 'ValueError'> (<class 'type'>)
```

Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).

```
Result (Buffer still empty): b'' (<class 'bytes'>)
Expectation (Buffer still empty): result = b'' (<class 'bytes'>)
```

Info	Processing wrong data (int)

Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).

```
Result (Wrong data exception): <class 'ValueError'> (<class 'type'>)
Expectation (Wrong data exception): result = <class 'ValueError'> (<class 'type'>)
```

Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).

```
Result (Buffer still empty): b'' (<class 'bytes'>)
Expectation (Buffer still empty): result = b'' (<class 'bytes'>)
```

Info	Processing wrong data (str)

Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).

```
Result (Wrong data exception): <class 'ValueError'> (<class 'type'>)
Expectation (Wrong data exception): result = <class 'ValueError'> (<class 'type'>)
```

Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).

```
Result (Buffer still empty): b'' (<class 'bytes'>)
Expectation (Buffer still empty): result = b'' (<class 'bytes'>)
```

A.1.13 REQ-0010

Testresult
This test was passed with the state: **Success**.

Info	Creating testframe for 'b'testframe for stp"

Success	STP-Frame is correct (Content b':<testframe for stp:>' and Type is <class 'bytes'>).

```
Result (STP-Frame): b':<testframe for stp:>' (<class 'bytes'>)
Expectation (STP-Frame): result = b':<testframe for stp:>' (<class 'bytes'>)
```

A.1.14 REQ-0015

Testresult
This test was passed with the state: **Success**.

Info	Creating testframe including start and end pattern for 'b'testframe for :<stp:>"

Success	STP-Frame is correct (Content b':<testframe for :=<stp:⇒:>' and Type is <class 'bytes'>).

```
Result (STP-Frame): b':<testframe for :=<stp:=>:>' (<class 'bytes'>)
Expectation (STP-Frame): result = b':<testframe for :=<stp:=>:>' (<class 'bytes'>)
```

A.1.15 REQ-0011

Testresult
This test was passed with the state: **Success**.

Info	Processing testframe: 'b':<testframe for stp:>"

```
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_1 ->
↪  STP_STATE_STORE_DATA
STP: data sync (3a) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2
STP: end pattern (3a 3e) received => storing message and changing state STP_STATE_ESCAPE_2 ->
↪  STP_STATE_IDLE
STP: message identified - (17): 74 65 73 74 66 72 61 6d 65 20 66 6f 72 20 73 74 70
```

Success	First processed STP snippet is correct (Content [] and Type is <class 'list'>).

```
Result (First processed STP snippet): [  ] (<class 'list'>)
Expectation (First processed STP snippet): result = [  ] (<class 'list'>)
```

Success	Final processed STP snippet is correct (Content [b'testframe for stp'] and Type is <class 'list'>).

```
Result (Final processed STP snippet): [ b'testframe for stp' ] (<class 'list'>)
Expectation (Final processed STP snippet): result = [ b'testframe for stp' ] (<class 'list'>)
```

A.1.16 REQ-0012

Testresult

This test was passed with the state: **Success**.

Info	Processing wrong data (list)

Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).

```
Result (Wrong data exception): <class 'ValueError'> (<class 'type'>)
Expectation (Wrong data exception): result = <class 'ValueError'> (<class 'type'>)
```

Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).

```
Result (Buffer still empty): b'' (<class 'bytes'>)
Expectation (Buffer still empty): result = b'' (<class 'bytes'>)
```

Info	Processing wrong data (int)

Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).

```
Result (Wrong data exception): <class 'ValueError'> (<class 'type'>)
Expectation (Wrong data exception): result = <class 'ValueError'> (<class 'type'>)
```

Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).

```
Result (Buffer still empty): b'' (<class 'bytes'>)
Expectation (Buffer still empty): result = b'' (<class 'bytes'>)
```

Info	Processing wrong data (str)

Success	Wrong data exception is correct (Content <class 'ValueError'> and Type is <class 'type'>).

```
Result (Wrong data exception): <class 'ValueError'> (<class 'type'>)
Expectation (Wrong data exception): result = <class 'ValueError'> (<class 'type'>)
```

Success	Buffer still empty is correct (Content b'' and Type is <class 'bytes'>).

```
Result (Buffer still empty): b'' (<class 'bytes'>)
Expectation (Buffer still empty): result = b'' (<class 'bytes'>)
```

A.1.17 REQ-0013

Testresult
This test was passed with the state: **Success**.

Info	Processing testframe: 'b':<testframe for :=<stp:⇒:>"

```
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_1 ->
↪  STP_STATE_STORE_DATA
STP: data sync (3a) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2
STP: store sync pattern (3a 3d) received => changing state STP_STATE_ESCAPE_2 ->
↪  STP_STATE_STORE_DATA
STP: data sync (3a) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2
STP: store sync pattern (3a 3d) received => changing state STP_STATE_ESCAPE_2 ->
↪  STP_STATE_STORE_DATA
STP: data sync (3a) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2
STP: end pattern (3a 3e) received => storing message and changing state STP_STATE_ESCAPE_2 ->
↪  STP_STATE_IDLE
STP: message identified - (21): 74 65 73 74 66 72 61 6d 65 20 66 6f 72 20 3a 3c 73 74 70 3a 3e
```

Success	Processed STP-Frame is correct (Content [b'testframe for :<stp:>'] and Type is <class 'list'>).

```
Result (Processed STP-Frame): [ b'testframe for :<stp:>' ] (<class 'list'>)
Expectation (Processed STP-Frame): result = [ b'testframe for :<stp:>' ] (<class 'list'>)
```

A.1.18 REQ-0014

Testresult
This test was passed with the state: **Success**.

Info	Processing testframe: 'b'_:<testframe for stp:>"

```
STP: no data sync (5f) received => ignoring byte
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_1 ->
↪  STP_STATE_STORE_DATA
STP: data sync (3a) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2
STP: end pattern (3a 3e) received => storing message and changing state STP_STATE_ESCAPE_2 ->
↪  STP_STATE_IDLE
STP: message identified - (17): 74 65 73 74 66 72 61 6d 65 20 66 6f 72 20 73 74 70
```

Success	Processed STP-Frame is correct (Content [b'testframe for stp'] and Type is <class 'list'>).

```
Result (Processed STP-Frame): [ b'testframe for stp' ] (<class 'list'>)
Expectation (Processed STP-Frame): result = [ b'testframe for stp' ] (<class 'list'>)
```

A.1.19 REQ-0016

Testresult
This test was passed with the state: **Success**.

Info	Processing data with an insufficient start pattern.

```
Sending b':1' to stp.
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: no start pattern (3a 31) received => changing state STP_STATE_ESCAPE_1 -> STP_STATE_IDLE
```

Success	Return value list if processing incorrect start of frame is correct (Content [[]] and Type is <class 'list'>).

```
Result (Return value list if processing incorrect start of frame): [ [  ] ] (<class 'list'>)
Expectation (Return value list if processing incorrect start of frame): result = [ [  ] ]
↪  (<class 'list'>)
```

Success	State after processing incorrect start of frame is correct (Content 0 and Type is <class 'int'>).

```
Result (State after processing incorrect start of frame): 0 (<class 'int'>)
Expectation (State after processing incorrect start of frame): result = 0 (<class 'int'>)
```

Info	Processing data with an insufficient start pattern (two times sync).

```
Sending b'::' to stp.
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: 2nd data sync (3a) received => keep state
```

Success	Return value list if processing data_sync twice is correct (Content [[]] and Type is <class 'list'>).

```
Result (Return value list if processing data_sync twice): [ [  ] ] (<class 'list'>)
Expectation (Return value list if processing data_sync twice): result = [ [  ] ] (<class
↪  'list'>)
```

Success	State after processing data_sync twice is correct (Content 1 and Type is <class 'int'>).

```
Result (State after processing data_sync twice): 1 (<class 'int'>)
Expectation (State after processing data_sync twice): result = 1 (<class 'int'>)
```

A.1.20 REQ-0017

Testresult
This test was passed with the state: **Success**.

Info	Processing data with an insufficient end pattern.

```
Sending b':<te:d' to stp.
```

```
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_1 ->
↪  STP_STATE_STORE_DATA
STP: data sync (3a) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2
STP: data (64) received => changing state STP_STATE_ESCAPE_2 -> STP_STATE_IDLE
STP: Chunking "(2): 74 65" from buffer
```

Success Return value list if processing data_sync and data again after start of frame is correct (Content [[]] and Type is <class 'list'>).

```
Result (Return value list if processing data_sync and data again after start of frame): [ [  ]
↪  ] (<class 'list'>)
Expectation (Return value list if processing data_sync and data again after start of frame):
↪  result = [ [  ] ] (<class 'list'>)
```

Success State after processing data_sync and data again after start of frame is correct (Content 0 and Type is <class 'int'>).

```
Result (State after processing data_sync and data again after start of frame): 0 (<class
↪  'int'>)
Expectation (State after processing data_sync and data again after start of frame): result = 0
↪  (<class 'int'>)
```

Success Buffer size after processing data with insufficient end pattern is correct (Content 0 and Type is <class 'int'>).

```
Result (Buffer size after processing data with insufficient end pattern): 0 (<class 'int'>)
Expectation (Buffer size after processing data with insufficient end pattern): result = 0
↪  (<class 'int'>)
```

Info Processing data with an insufficient end pattern (start pattern instead of end pattern).

```
Sending b':<te:<' to stp.
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_1 ->
↪  STP_STATE_STORE_DATA
STP: data sync (3a) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_2 ->
↪  STP_STATE_STORE_DATA
STP: Chunking "(2): 74 65" from buffer
```

Success Return value list if processing 2nd start of frame is correct (Content [[]] and Type is <class 'list'>).

```
Result (Return value list if processing 2nd start of frame): [ [  ] ] (<class 'list'>)
Expectation (Return value list if processing 2nd start of frame): result = [ [  ] ] (<class
↪  'list'>)
```

Success State after processing 2nd start of frame is correct (Content 3 and Type is <class 'int'>).

```
Result (State after processing 2nd start of frame): 3 (<class 'int'>)
Expectation (State after processing 2nd start of frame): result = 3 (<class 'int'>)
```

Success Buffer size after processing 2nd start of frame is correct (Content 0 and Type is <class 'int'>).

```
Result (Buffer size after processing 2nd start of frame): 0 (<class 'int'>)
Expectation (Buffer size after processing 2nd start of frame): result = 0 (<class 'int'>)
```

Info Processing data with an insufficient end pattern (two times sync instead of end pattern).

```
Sending b':<te::' to stp.
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_1 ->
↪  STP_STATE_STORE_DATA
STP: data sync (3a) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2
STP: second data sync (3a) received => changing state STP_STATE_ESCAPE_2 -> STP_STATE_ESCAPE_1
STP: Chunking "(2): 74 65" from buffer
```

Success Return value list if processing data_sync twice after start of frame is correct (Content [[]] and Type is <class 'list'>).

```
Result (Return value list if processing data_sync twice after start of frame): [ [  ] ]
↪  (<class 'list'>)
Expectation (Return value list if processing data_sync twice after start of frame): result = [
↪  [  ] ] (<class 'list'>)
```

Success State after processing data_sync twice after start of frame is correct (Content 1 and Type is <class 'int'>).

```
Result (State after processing data_sync twice after start of frame): 1 (<class 'int'>)
Expectation (State after processing data_sync twice after start of frame): result = 1 (<class
↪  'int'>)
```

A.1.21 REQ-0018

Testresult

This test was passed with the state: **Success**.

Info Corrupting stp state and processing data.

```
Sending b':<te' to stp.
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
```

```
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_1 ->
↪  STP_STATE_STORE_DATA
Setting state of stp to 255.
Sending b':<te' to stp.
STP: unknown state (255) => adding value (3a) back to data again and changing state ->
↪  STP_STATE_IDLE
STP: Chunking "(2): 74 65" from buffer
STP: data sync (3a) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1
STP: start pattern (3a 3c) received => changing state STP_STATE_ESCAPE_1 ->
↪  STP_STATE_STORE_DATA
```

Success	Return value list if processing start of a frame after state had been corrupted is correct (Content [[]] and Type is <class 'list'>).

```
Result (Return value list if processing start of a frame after state had been corrupted): [ [
↪  ] ] (<class 'list'>)
Expectation (Return value list if processing start of a frame after state had been corrupted):
↪  result = [ [  ] ] (<class 'list'>)
```

Success	State after processing start of a frame after state had been corrupted is correct (Content 3 and Type is <class 'int'>).

```
Result (State after processing start of a frame after state had been corrupted): 3 (<class
↪  'int'>)
Expectation (State after processing start of a frame after state had been corrupted): result =
↪  3 (<class 'int'>)
```

Success	Buffer size after corrupting stp state is correct (Content 2 and Type is <class 'int'>).

```
Result (Buffer size after corrupting stp state): 2 (<class 'int'>)
Expectation (Buffer size after corrupting stp state): result = 2 (<class 'int'>)
```

B Test-Coverage

B.1 stringtools

The line coverage for stringtools was 100.0%
The branch coverage for stringtools was 96.9%

B.1.1 stringtools.__init__.py

The line coverage for stringtools.__init__.py was 100.0%
The branch coverage for stringtools.__init__.py was 96.9%

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
"""
stringtools (Stringtools)
=========================

**Author:**

* Dirk Alders <sudo-dirk@mount-mockery.de>

**Description:**

    This Module supports functionality around string operations.

**Submodules:**

* :mod:`stringtools.csp`
* :mod:`stringtools.stp`
* :func:`gzip_compress`
* :func:`gzip_extract`
* :func:`hexlify`

**Unittest:**

        See also the :download:`unittest <stringtools/_testresults_/unittest.pdf>` documentation.

**Module Documentation:**

"""

from stringtools import stp
from stringtools import csp
__DEPENDENCIES__ = []

import fractions
import gzip
import logging
import time
import sys

try:
    from config import APP_NAME as ROOT_LOGGER_NAME
except ImportError:
    ROOT_LOGGER_NAME = 'root'
logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)

__DESCRIPTION__ = """The Module {\\tt %s} is designed to support functionality for strings (e.g.
    transfer strings via a bytestream, compressing, extracting, ...).
For more Information read the sphinx documentation.""" % __name__.replace('_', '\\_')
"""The Module Description"""
__INTERPRETER__ = (3, )
"""The Tested Interpreter-Versions"""

__all__ = ['gzip_compress',
           'gzip_extract',
           'hexlify',
           'csp',
           'stp']


```

```
def physical_value_repr(value, unit=''):
    prefix = {
        -4: 'p',
        -3: 'n',
        -2: 'u',
        -1: 'm',
        0: '',
        1: 'k',
        2: 'M',
        3: 'G',
        4: 'T',
        5: 'P',
    }
    u = 0
    while u > -4 and u < 5 and (value >= 1000. or value < 1.) and value != 0:
        if value >= 1:
            u += 1
            value /= 1000.
        else:
            u -= 1
            value *= 1000.
    if u == 0:
        ext = ''
    else:
        ext = prefix[u]
    #
    if value < 100.:
        value = '%.2f' % (value)
    else:
        value = '%.1f' % (value)
    while value.find('.') > -1 and (value.endswith('0') or value.endswith('.')):
        value = value[:-1]
    return value + ext + unit


def time_repr(seconds):
    days = seconds / (24 * 60 * 60)
    seconds = seconds % (24 * 60 * 60)
    if seconds >= 60 * 60:
        rv = time.strftime('%H:%M:%S', time.gmtime(seconds))
    else:
        rv = time.strftime('%M:%S', time.gmtime(seconds))
    if days >= 1:
        rv = '%dD %s' % (days, rv)
    if rv.endswith(' 00:00'):
        rv = rv[:-6]
    return rv


def frac_repr(value):
    f = fractions.Fraction(value).limit_denominator()
    return '%s/%s' % (f.numerator, f.denominator)


def gzip_compress(s, compresslevel=9):
    """
    Method to compress a stream of bytes.

    :param str s: The bytestream (string) to be compressed
    :param int compresslevel: An optional compressionn level (default is 9)
    :return: The compressed bytestream
    :rtype: str
```

```
123
124     **Example:**
125
126     .. literalinclude:: stringtools/_examples_/gzip_compress.py
127
128     Will result to the following output:
129
130     .. literalinclude:: stringtools/_examples_/gzip_compress.log
131     """
132     rv = None
133     t = time.time()
134     rv = gzip.compress(s, compresslevel)
135     if rv is not None:
136         logger.debug('GZIP: Finished to compress a string (compression_rate=%.3f, consumed_time
    =%.1fs).', len(rv) / float(len(s)), time.time() - t)
137     return rv
138
139
140 def gzip_extract(s):
141     """
142     Method to extract data from a compress stream of bytes.
143
144     :param str s: The compressed bytestream (string) to be extracted
145     :return: The extracted data
146     :rtype: str
147
148     **Example:**
149
150     .. literalinclude:: stringtools/_examples_/gzip_extract.py
151
152     Will result to the following output:
153
154     .. literalinclude:: stringtools/_examples_/gzip_extract.log
155     """
156     t = time.time()
157     rv = None
158     rv = gzip.decompress(s)
159     if rv is not None:
160         logger.debug('GZIP: Finished to extract a string (compression_rate=%.3f, consumed_time
    =%.1fs).', len(s) / float(len(rv)), time.time() - t)
161     return rv
162
163
164 def hexlify(s):
165     """Method to hexlify a string.
166
167     :param str s: A string including the bytes to be hexlified.
168     :returns: The hexlified string
169     :rtype: str
170
171     **Example:**
172
173     .. literalinclude:: stringtools/_examples_/hexlify.py
174
175     Will result to the following output:
176
177     .. literalinclude:: stringtools/_examples_/hexlify.log
178     """
179     rv = '(%d):' % len(s)
180     for byte in s:
181         rv += ' %02x' % byte
182     return rv
```

```
183
184
185 def linefeed_filter(s):
186     """Method to change linefeed and carriage return to '\\\\n' and '\\\\r'
187
188     :param str s: A string including carriage return and/ or linefeed.
189     :returns: A string with converted carriage return and/ or linefeed.
190     :rtype: str
191     """
192     return s.replace(b'\r', b'\\r').replace(b'\n', b'\\n')
```

B.1.2 stringtools.csp.py

The line coverage for stringtools.csp.py was 100.0%
The branch coverage for stringtools.csp.py was 96.9%

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #
4 """
5 stringtools.csp (Carriage-Return seperation protocol)
6 ====================================================
7
8 **Author:**
9
10 * Dirk Alders <sudo-dirk@mount-mockery.de>
11
12 **Description:**
13
14     This module is a submodule of :mod:`stringtools` and creates an frame to transmit and receive
       messages via an serial interface.
15
16 **Submodules:**
17
18 * :class:`stringtools.csp.csp`
19 * :func:`stringtools.csp.build_frame`
20 """
21
22 import stringtools
23
24 import logging
25 import sys
26
27 try:
28     from config import APP_NAME as ROOT_LOGGER_NAME
29 except ImportError:
30     ROOT_LOGGER_NAME = 'root'
31 logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)
32
33 DATA_SEPERATOR = b'\n'
34
35
36 class csp(object):
37     """This class extracts messages from an "csp-stream".
38
39     **Example:**
40
41     .. literalinclude:: stringtools/_examples_/csp.csp.py
42
43     Will result to the following output:
44
45     .. literalinclude:: stringtools/_examples_/csp.csp.log
46     """
```

```
47      LOG_PREFIX = 'CSP:'
48
49      def __init__(self, seperator=DATA_SEPERATOR):
50          self.__buffer__ = b''
51          self.__seperator__ = seperator
52
53      def process(self, data):
54          """
55          This processes a byte out of a "stp-stream".
56
57          :param bytes data: A byte stream
58          :returns: A list of the extracted message(s)
59          :rtype: list
60          """
61          rv = (self.__buffer__ + data).split(self.__seperator__)
62          self.__buffer__ = rv.pop()
63          if len(self.__buffer__) != 0:
64              logger.debug('%s Leaving data in buffer (to be processed next time): %s', self.
    LOG_PREFIX, stringtools.hexlify(self.__buffer__))
65          for msg in rv:
66              logger.info('%s message identified - %s', self.LOG_PREFIX, stringtools.hexlify(msg))
67          return rv
68
69
70  def build_frame(msg, seperator=DATA_SEPERATOR):
71      """This Method builds an "csp-frame" to be transfered via a stream.
72
73      :param str data: A String (Bytes) to be framed
74      :returns: The "csp-framed" message to be sent
75      :rtype: str
76
77      **Example:**
78
79      .. literalinclude:: stringtools/_examples_/csp.build_frame.py
80
81      Will result to the following output:
82
83      .. literalinclude:: stringtools/_examples_/csp.build_frame.log
84      """
85      if seperator in msg:
86          raise ValueError
87      else:
88          return msg + seperator
```

B.1.3 stringtools.stp.py

The line coverage for stringtools.stp.py was 100.0%
The branch coverage for stringtools.stp.py was 96.9%

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  #
4  """
5  stringtools.stp (Serial transfer protocol)
6  ==========================================
7
8  **Author:**
9
10 * Dirk Alders <sudo-dirk@mount-mockery.de>
```

```
11
12 **Description:**
13
14     This module is a submodule of :mod:`stringtools` and creates an serial frame to transmit and
       receive messages via an serial interface.
15
16 **Submodules:**
17
18 * :class:`stringtools.stp.stp`
19 * :func:`stringtools.stp.build_frame`
20 """
21
22 import stringtools
23
24 import logging
25 import sys
26
27 try:
28     from config import APP_NAME as ROOT_LOGGER_NAME
29 except ImportError:
30     ROOT_LOGGER_NAME = 'root'
31 logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)
32
33 DATA_SYNC = b'\x3a'
34 """The data sync byte"""
35 DATA_CLEAR_BUFFER = b'\x3c'
36 """The clear buffer byte ('\\\\x3a\\\\x3c' -> start of message)"""
37 DATA_VALID_MSG = b'\x3e'
38 """The valid message byte ('\\\\x3a\\\\x3e' -> end of message)"""
39 DATA_STORE_SYNC_VALUE = b'\x3d'
40 """The store sync value byte ('\\\\x3a\\\\x3d' -> '\\\\x3a' inside a message)"""
41
42 STP_STATE_IDLE = 0x00
43 """Idle state definition (default)"""
44 STP_STATE_ESCAPE_1 = 0x01
45 """Escape 1 state definition ('\\\\x3a\\\\x3c' found)"""
46 STP_STATE_ESCAPE_2 = 0x02
47 """Escape 2 state definition ('\\\\x3a' found inside a message)"""
48 STP_STATE_STORE_DATA = 0x03
49 """Store data state definition (start of message found; data will be stored)"""
50
51
52 class stp(object):
53     """This class extracts messages from an "stp-stream".
54
55     **Example:**
56
57     .. literalinclude:: stringtools/_examples_/stp.stp.py
58
59     Will result to the following output:
60
61     .. literalinclude:: stringtools/_examples_/stp.stp.log
62     """
63     LOG_PREFIX = 'STP:'
64
65     def __init__(self):
66         self.state = STP_STATE_IDLE
67         self.__buffer__ = b''
68         self.__clear_buffer__()
69
```

```
70      def __clear_buffer__(self):
71          if len(self.__buffer__) > 0:
72              logger.warning('%s Chunking "%s" from buffer', self.LOG_PREFIX, stringtools.hexlify(
    self.__buffer__))
73          self.__buffer__ = b''
74
75      def process(self, data):
76          """
77          This processes a byte out of a "stp-stream".
78
79          :param bytes data: A byte stream
80          :returns: The extracted message or None, if no message is identified yet
81          :rtype: str
82          """
83          if type(data) is list:
84              raise TypeError
85          #
86          rv = []
87          #
88          while len(data) > 0:
89              b = bytes([data[0]])
90              data = data[1:]
91              #
92              if self.state == STP_STATE_IDLE:
93                  if b == DATA_SYNC:
94                      self.state = STP_STATE_ESCAPE_1
95                      logger.debug('%s data sync (%02x) received => changing state STP_STATE_IDLE
    -> STP_STATE_ESCAPE_1', self.LOG_PREFIX, ord(b))
96                  else:
97                      logger.warning('%s no data sync (%02x) received => ignoring byte', self.
    LOG_PREFIX, ord(b))
98              elif self.state == STP_STATE_ESCAPE_1:
99                  if b == DATA_CLEAR_BUFFER:
100                     logger.debug('%s start pattern (%02x %02x) received => changing state
    STP_STATE_ESCAPE_1 -> STP_STATE_STORE_DATA', self.LOG_PREFIX, ord(DATA_SYNC), ord(b))
101                     self.state = STP_STATE_STORE_DATA
102                     self.__clear_buffer__()
103                 elif b != DATA_SYNC:
104                     self.state = STP_STATE_IDLE
105                     logger.warning('%s no start pattern (%02x %02x) received => changing state
    STP_STATE_ESCAPE_1 -> STP_STATE_IDLE', self.LOG_PREFIX, ord(DATA_SYNC), ord(b))
106                 else:
107                     logger.warning('%s 2nd data sync (%02x) received => keep state', self.
    LOG_PREFIX, ord(b))
108             elif self.state == STP_STATE_STORE_DATA:
109                 if b == DATA_SYNC:
110                     self.state = STP_STATE_ESCAPE_2
111                     logger.debug('%s data sync (%02x) received => changing state
    STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2', self.LOG_PREFIX, ord(b))
112                 else:
113                     self.__buffer__ += b
114             elif self.state == STP_STATE_ESCAPE_2:
115                 if b == DATA_CLEAR_BUFFER:
116                     logger.warning('%s start pattern (%02x %02x) received => changing state
    STP_STATE_ESCAPE_2 -> STP_STATE_STORE_DATA', self.LOG_PREFIX, ord(DATA_SYNC), ord(b))
117                     self.state = STP_STATE_STORE_DATA
118                     self.__clear_buffer__()
119                 elif b == DATA_VALID_MSG:
120                     self.state = STP_STATE_IDLE
121                     logger.debug('%s end pattern (%02x %02x) received => storing message and
    changing state STP_STATE_ESCAPE_2 -> STP_STATE_IDLE', self.LOG_PREFIX, ord(DATA_SYNC), ord(b)
    )
```

```
                    rv.append(self.__buffer__)
                    self.__buffer__ = b''
                elif b == DATA_STORE_SYNC_VALUE:
                    self.state = STP_STATE_STORE_DATA
                    logger.debug('%s store sync pattern (%02x %02x) received => changing state STP_STATE_ESCAPE_2 -> STP_STATE_STORE_DATA', self.LOG_PREFIX, ord(DATA_SYNC), ord(b))
                    self.__buffer__ += DATA_SYNC
                elif b == DATA_SYNC:
                    self.state = STP_STATE_ESCAPE_1
                    logger.warning('%s second data sync (%02x) received => changing state STP_STATE_ESCAPE_2 -> STP_STATE_ESCAPE_1', self.LOG_PREFIX, ord(b))
                    self.__clear_buffer__()
                else:
                    self.state = STP_STATE_IDLE
                    logger.warning('%s data (%02x) received => changing state STP_STATE_ESCAPE_2 -> STP_STATE_IDLE', self.LOG_PREFIX, ord(b))
                    self.__clear_buffer__()
            else:
                logger.error('%s unknown state (%s) => adding value (%02x) back to data again and changing state -> STP_STATE_IDLE', self.LOG_PREFIX, repr(self.state), ord(b))
                self.state = STP_STATE_IDLE
                self.__clear_buffer__()
                data = b + data
        for msg in rv:
            logger.info('%s message identified - %s', self.LOG_PREFIX, stringtools.hexlify(msg))
        return rv


def build_frame(data):
    """This Method builds an "stp-frame" to be transfered via a stream.

    :param str data: A String (Bytes) to be framed
    :returns: The "stp-framed" message to be sent
    :rtype: str

    **Example:**

    .. literalinclude:: stringtools/_examples_/stp.build_frame.py

    Will result to the following output:

    .. literalinclude:: stringtools/_examples_/stp.build_frame.log
    """
    rv = DATA_SYNC + DATA_CLEAR_BUFFER

    for byte in data:
        byte = bytes([byte])
        if byte == DATA_SYNC:
            rv += DATA_SYNC + DATA_STORE_SYNC_VALUE
        else:
            rv += byte

    rv += DATA_SYNC + DATA_VALID_MSG
    return rv
```