Unittest for `stringtools`

February 3, 2020

Contents

1 Test Information 4

1.1 Test Candidate Information . . . 4

1.2 Unittest Information . . . 4

1.3 Test System Information . . . 4

2 Statistic 4

2.1 Test-Statistic for testrun with python 2.7.17 (final) . . . 4

2.2 Test-Statistic for testrun with python 3.6.9 (final) . . . 5

2.3 Coverage Statistic . . . 5

3 Tested Requirements 6

3.1 Stream Definition . . . 6

3.1.1 Physical representation . . . 6

3.1.2 Time representation . . . 7

3.1.3 Fraction representation . . . 8

3.2 Human readable value representations . . . 9

3.3 Stream to Human readable String . . . 9

3.3.1 Hexadecimal Values . . . 9

3.3.2 Number of Bytes . . . 10

3.3.3 CRLF-Filter . . . 10

3.4 Stream Compression . . . 11

3.4.1 Compress . . . 11

3.4.2 Extract . . . 13

3.5 Carriagereturn Seperation Protocol (CSP) . . . 14

3.5.1 Frame creation . . . 14

3.5.2 Frame creation error . . . 16

3.5.3 Frame processing . . . 17

3.5.4 Frame processing - Input data type error . . . 18

3.6 Serial Transfer Protocol (STP) . . . 19

3.6.1 Frame creation . . . 19

3.6.2 Frame creation - Start pattern and end pattern inside a message 20
3.6.3 Frame processing 21
3.6.4 Frame processing - Input data type error 22
3.6.5 Frame processing - Start pattern and end pattern inside a message 23
3.6.6 Frame processing - Data before the start pattern 23
3.6.7 Frame processing - Incorrect start patterns 24
3.6.8 Frame processing - Incorrect end pattern 25
3.6.9 Frame processing - After state corruption 26

A Trace for testrun with python 2.7.17 (final) 28

A.1 Tests with status Info (21) 28
A.1.1 Physical representation 28
A.1.2 Time representation 29
A.1.3 Fraction representation 30
A.1.4 Hexadecimal Values 31
A.1.5 Number of Bytes 32
A.1.6 CRLF-Filter 32
A.1.7 Compress 32
A.1.8 Extract 33
A.1.9 Frame creation 34
A.1.10 Frame creation error 34
A.1.11 Frame processing 35
A.1.12 Frame processing - Input data type error 36
A.1.13 Frame creation 37
A.1.14 Frame creation - Start pattern and end pattern inside a message 37
A.1.15 Frame processing 38
A.1.16 Frame processing - Input data type error 39
A.1.17 Frame processing - Start pattern and end pattern inside a message 40
A.1.18 Frame processing - Data before the start pattern 40
A.1.19 Frame processing - Incorrect start patterns 41
A.1.20 Frame processing - Incorrect end pattern 42
A.1.21 Frame processing - After state corruption 44

B Trace for testrun with python 3.6.9 (final) 45

B.1 Tests with status Info (21) 45

B.1.1 Physical representation 45

B.1.2 Time representation 47

B.1.3 Fraction representation 48

B.1.4 Hexadecimal Values 49

B.1.5 Number of Bytes 49

B.1.6 CRLF-Filter 50

B.1.7 Compress 50

B.1.8 Extract 51

B.1.9 Frame creation 52

B.1.10 Frame creation error 52

B.1.11 Frame processing 53

B.1.12 Frame processing - Input data type error 53

B.1.13 Frame creation 55

B.1.14 Frame creation - Start pattern and end pattern inside a message 55

B.1.15 Frame processing 56

B.1.16 Frame processing - Input data type error 56

B.1.17 Frame processing - Start pattern and end pattern inside a message 58

B.1.18 Frame processing - Data before the start pattern 58

B.1.19 Frame processing - Incorrect start patterns 59

B.1.20 Frame processing - Incorrect end pattern 60

B.1.21 Frame processing - After state corruption 62

C Test-Coverage 63

C.1 `stringtools` 63

C.1.1 `stringtools.__init__.py` 64

C.1.2 `stringtools.csp.py` 67

C.1.3 `stringtools.stp.py` 69

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	python2, python3
Version	8a325608d13645870f6f32774828606f
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	50d96e3d37f4672ae07c76a1a12c6546
Testruns with	python 2.7.17 (final), python 3.6.9 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	LinuxMint 19.3 tricia
Hostname	ahorn
Kernel	5.3.0-28-generic (#30 18.04.1-Ubuntu SMP Fri Jan 17 06:14:09 UTC 2020)
Machine	x86_64
Path	/user_data/data/dirk/prj/unittest/stringtools/unittest
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun with python 2.7.17 (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	0.020s

2.2 Test-Statistic for testrun with python 3.6.9 (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	0.016s

2.3 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
`stringtools`	100.0%	97.7%
`stringtools.__init__.py`	100.0%	
`stringtools.csp.py`	100.0%	
`stringtools.stp.py`	100.0%	

3 Tested Requirements

3.1 Stream Definition

A Stream is from class `bytes` for python3 and from type `str` for python2.

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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (24)
Start-Time:	2020-02-03 18:42:08,358
Finished-Time:	2020-02-03 18:42:08,361
Time-Consumption	0.002s
Testsummary:	
Success	Physical representation for 1.17e-10 is correct (Content '117p' and Type is <type 'str'>).
Success	Physical representation for 5.4e-08 is correct (Content '54n' and Type is <type 'str'>).
Success	Physical representation for 2.53e-05 is correct (Content '25.3/xc2/xb5' and Type is <type 'str'>).
Success	Physical representation for 0.1 is correct (Content '100m' and Type is <type 'str'>).
Success	Physical representation for 1 is correct (Content '1' and Type is <type 'str'>).
Success	Physical representation for 1000 is correct (Content '1k' and Type is <type 'str'>).
Success	Physical representation for 1005001 is correct (Content '1.01M' and Type is <type 'str'>).
Success	Physical representation for 1004000000 is correct (Content '1G' and Type is <type 'str'>).
Success	Physical representation for 1003000000000 is correct (Content '1T' and Type is <type 'str'>).
Success	Physical representation for 10000000000000000 is correct (Content '10P' and Type is <type 'str'>).
Success	Physical representation for 17.17 is correct (Content '17.17' and Type is <type 'str'>).
Success	Physical representation for 117000 is correct (Content '117k' and Type is <type 'str'>).
Success	Physical representation for 117.17 is correct (Content '117.2' and Type is <type 'str'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (24)
Start-Time:	2020-02-03 18:42:08,767
Finished-Time:	2020-02-03 18:42:08,769
Time-Consumption	0.002s

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 '25.3' and Type is <class 'str'>).
Success	Physical representation for 0.1 is correct (Content '100m' 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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (25)
Start-Time:	2020-02-03 18:42:08,361
Finished-Time:	2020-02-03 18:42:08,362
Time-Consumption	0.001s

Testsummary:

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

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (25)

Start-Time:	2020-02-03 18:42:08,770
Finished-Time:	2020-02-03 18:42:08,771
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (26)
Start-Time:	2020-02-03 18:42:08,362
Finished-Time:	2020-02-03 18:42:08,363
Time-Consumption	0.001s

Testsummary:

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

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (26)
Start-Time:	2020-02-03 18:42:08,771
Finished-Time:	2020-02-03 18:42:08,772
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (29)
Start-Time:	2020-02-03 18:42:08,363
Finished-Time:	2020-02-03 18:42:08,364
Time-Consumption	0.000s
Testsummary:	
Info	Checking test pattern de ad be ef (<type 'str'>).
Success	Pattern included all relevant information in the correct order.

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (29)
Start-Time:	2020-02-03 18:42:08,772
Finished-Time:	2020-02-03 18:42:08,772
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (30)
Start-Time:	2020-02-03 18:42:08,364
Finished-Time:	2020-02-03 18:42:08,364
Time-Consumption	0.000s
Testsummary:	
Info	Checking test pattern with length 4.
Success	'(4)' is in '(4): de ad be ef' at position 0

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (30)
Start-Time:	2020-02-03 18:42:08,772
Finished-Time:	2020-02-03 18:42:08,772
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (31)
Start-Time:	2020-02-03 18:42:08,364
Finished-Time:	2020-02-03 18:42:08,364
Time-Consumption	0.000s
Testsummary:	
Info	Checking test pattern with length 4.
Success	Returnvalue of linefeed_filter is correct (Content 'test//r//n123//r//n' and Type is <type 'str'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (31)
Start-Time:	2020-02-03 18:42:08,773
Finished-Time:	2020-02-03 18:42:08,773
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (34)
Start-Time:	2020-02-03 18:42:08,364
Finished-Time:	2020-02-03 18:42:08,365
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 ff ff
Info	Extracting stream: (26): 1f 8b 08 00 70 5b 38 5e 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 ff and Type is <type 'str'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (34)
Start-Time:	2020-02-03 18:42:08,773
Finished-Time:	2020-02-03 18:42:08,774
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 ff ff
Info	Extracting stream: (26): 1f 8b 08 00 70 5b 38 5e 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 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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (35)
Start-Time:	2020-02-03 18:42:08,365
Finished-Time:	2020-02-03 18:42:08,366
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 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff' and Type is <type 'str'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (35)
Start-Time:	2020-02-03 18:42:08,774
Finished-Time:	2020-02-03 18:42:08,774
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 ff ff 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)

The Carriagereturn Seperation Protocol shall use carriage return as the end pattern for message seperation.

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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (39)
Start-Time:	2020-02-03 18:42:08,366
Finished-Time:	2020-02-03 18:42:08,366
Time-Consumption	0.000s
Testsummary:	
Info	Creating testframe for ":testframe: for csp"
Success	CSP-Frame is correct (Content ':testframe: for csp/n' and Type is <type 'str'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (39)
Start-Time:	2020-02-03 18:42:08,774
Finished-Time:	2020-02-03 18:42:08,775
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (40)
Start-Time:	2020-02-03 18:42:08,366
Finished-Time:	2020-02-03 18:42:08,366
Time-Consumption	0.000s
Testsummary:	
Info	Creating testframe for ":testframe: for csp"
Success	CSP-Frame is correct (Content <type 'exceptions.ValueError'> and Type is <type 'type'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (40)
Start-Time:	2020-02-03 18:42:08,775
Finished-Time:	2020-02-03 18:42:08,775
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (41)
Start-Time:	2020-02-03 18:42:08,367
Finished-Time:	2020-02-03 18:42:08,367
Time-Consumption	0.001s
Testsummary:	
Info	Processing testframe: ":testframe: for csp/n" in two snippets
Success	First processed CSP-Snippet is correct (Content [] and Type is <type 'list'>).
Success	Final processed CSP-Frame is correct (Content [':testframe: for csp'] and Type is <type 'list'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (41)
Start-Time:	2020-02-03 18:42:08,775
Finished-Time:	2020-02-03 18:42:08,776

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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (42)
Start-Time:	2020-02-03 18:42:08,367
Finished-Time:	2020-02-03 18:42:08,369
Time-Consumption	0.001s
Testsummary:	
Info	Processing wrong data (list)
Success	Wrong data exception is correct (Content <type 'exceptions.ValueError'> and Type is <type 'type'>).
Success	Buffer still empty is correct (Content " and Type is <type 'str'>).
Info	Processing wrong data (int)
Success	Wrong data exception is correct (Content <type 'exceptions.ValueError'> and Type is <type 'type'>).
Success	Buffer still empty is correct (Content " and Type is <type 'str'>).
Info	Processing wrong data (unicode)
Success	Wrong data exception is correct (Content <type 'exceptions.ValueError'> and Type is <type 'type'>).
Success	Buffer still empty is correct (Content " and Type is <type 'str'>).

Testresult

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

Testrun:	python 3.6.9 (final)

Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (42)
Start-Time:	2020-02-03 18:42:08,776
Finished-Time:	2020-02-03 18:42:08,778
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)

The Serial Transfer Protocol shall use a start pattern and an end pattern to identify a message in a stream. Both patterns shall be a two byte values starting with the same (sync-)byte.

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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (46)
Start-Time:	2020-02-03 18:42:08,369
Finished-Time:	2020-02-03 18:42:08,369
Time-Consumption	0.000s

Testsummary:

Info	Creating testframe for "testframe for stp"
Success	STP-Frame is correct (Content ':<testframe for stp:>' and Type is <type 'str'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (46)
Start-Time:	2020-02-03 18:42:08,778
Finished-Time:	2020-02-03 18:42:08,778
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (47)
Start-Time:	2020-02-03 18:42:08,369
Finished-Time:	2020-02-03 18:42:08,369
Time-Consumption	0.000s
Testsummary:	
Info	Creating testframe including start and end pattern for "testframe for :<stp:>"
Success	STP-Frame is correct (Content ':<testframe for :=<stp:=>:>' and Type is <type 'str'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (47)
Start-Time:	2020-02-03 18:42:08,778
Finished-Time:	2020-02-03 18:42:08,779
Time-Consumption	0.001s

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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (48)
Start-Time:	2020-02-03 18:42:08,370
Finished-Time:	2020-02-03 18:42:08,371
Time-Consumption	0.001s
Testsummary:	
Info	Processing testframe: ":<testframe for stp:>"
Success	First processed STP snippet is correct (Content [] and Type is <type 'list'>).
Success	Final processed STP snippet is correct (Content ['testframe for stp'] and Type is <type 'list'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (48)
Start-Time:	2020-02-03 18:42:08,779
Finished-Time:	2020-02-03 18:42:08,780
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (49)
Start-Time:	2020-02-03 18:42:08,371
Finished-Time:	2020-02-03 18:42:08,374
Time-Consumption	0.003s
Testsummary:	
Info	Processing wrong data (list)
Success	Wrong data exception is correct (Content <type 'exceptions.ValueError'> and Type is <type 'type'>).
Success	Buffer still empty is correct (Content '' and Type is <type 'str'>).
Info	Processing wrong data (int)
Success	Wrong data exception is correct (Content <type 'exceptions.ValueError'> and Type is <type 'type'>).
Success	Buffer still empty is correct (Content '' and Type is <type 'str'>).
Info	Processing wrong data (unicode)
Success	Wrong data exception is correct (Content <type 'exceptions.ValueError'> and Type is <type 'type'>).
Success	Buffer still empty is correct (Content '' and Type is <type 'str'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (49)
Start-Time:	2020-02-03 18:42:08,780
Finished-Time:	2020-02-03 18:42:08,781
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

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

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

Testrun:	python 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (50)
Start-Time:	2020-02-03 18:42:08,374
Finished-Time:	2020-02-03 18:42:08,375
Time-Consumption	0.001s
Testsummary:	
Info	Processing testframe: '':<testframe for :=<stp:⇒:>''
Success	Processed STP-Frame is correct (Content ['testframe for :<stp:>'] and Type is <type 'list'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (50)
Start-Time:	2020-02-03 18:42:08,781
Finished-Time:	2020-02-03 18:42:08,782
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (51)
Start-Time:	2020-02-03 18:42:08,375
Finished-Time:	2020-02-03 18:42:08,376
Time-Consumption	0.001s
Testsummary:	
Info	Processing testframe: "_:<testframe for stp:>"
Success	Processed STP-Frame is correct (Content ['testframe for stp'] and Type is <type 'list'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (51)
Start-Time:	2020-02-03 18:42:08,782
Finished-Time:	2020-02-03 18:42:08,782
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (52)
Start-Time:	2020-02-03 18:42:08,376
Finished-Time:	2020-02-03 18:42:08,377
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 <type 'list'>).
Success	State after processing incorrect start of frame is correct (Content 0 and Type is <type '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 <type 'list'>).
Success	State after processing data_sync twice is correct (Content 1 and Type is <type 'int'>).

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (52)
Start-Time:	2020-02-03 18:42:08,782
Finished-Time:	2020-02-03 18:42:08,783
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (53)
Start-Time:	2020-02-03 18:42:08,377
Finished-Time:	2020-02-03 18:42:08,380
Time-Consumption	0.003s
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 <type 'list'>).
Success	State after processing data_sync and data again after start of frame is correct (Content 0 and Type is <type 'int'>).
Success	Buffer size after processing data with insufficient end pattern is correct (Content 0 and Type is <type '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 <type 'list'>).
Success	State after processing 2nd start of frame is correct (Content 3 and Type is <type 'int'>).
Success	Buffer size after processing 2nd start of frame is correct (Content 0 and Type is <type '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 <type 'list'>).
Success	State after processing data_sync twice after start of frame is correct (Content 1 and Type is <type 'int'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (53)
Start-Time:	2020-02-03 18:42:08,783
Finished-Time:	2020-02-03 18:42:08,785
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 2.7.17 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (54)
Start-Time:	2020-02-03 18:42:08,380
Finished-Time:	2020-02-03 18:42:08,381
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 <type 'list'>).
Success	State after processing start of a frame after state had been corrupted is correct (Content 3 and Type is <type 'int'>).
Success	Buffer size after corrupting stp state is correct (Content 2 and Type is <type 'int'>).

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (54)
Start-Time:	2020-02-03 18:42:08,785
Finished-Time:	2020-02-03 18:42:08,786
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 2.7.17 (final)

A.1 Tests with status Info (21)

A.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**.

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

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

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

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

Success	Physical representation for 2.53e-05 is correct (Content '25.3/xc2/xb5' and Type is <type 'str'>).

```
Result (Physical representation for 2.53e-05): '25.3\xc2\xb5' (<type 'str'>)
Expectation (Physical representation for 2.53e-05): result = '25.3\xc2\xb5' (<type 'str'>)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A.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**.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A.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**.

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

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

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

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

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

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

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

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

A.1.4 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**.

Info	Checking test pattern de ad be ef (<type 'str'>).

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 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**.

Info	Checking test pattern with length 4.

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

A.1.6 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**.

Info	Checking test pattern with length 4.

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

```
Result (Returnvalue of linefeed_filter): 'test\\r\\n123\\r\\n' (<type 'str'>)
Expectation (Returnvalue of linefeed_filter): result = 'test\\r\\n123\\r\\n' (<type 'str'>)
```

A.1.7 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**.

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 ff ff

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

Info	Extracting stream: (26): 1f 8b 08 00 70 5b 38 5e 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 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff and Type is <type 'str'>).

```
Result (Extracted data): (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 ff (<type 'str'>)
```

```
Expectation (Extracted data): result = (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 ff (<type 'str'>)
```

A.1.8 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**.

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 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff' and Type is <type 'str'>).

```
Result (Extracted data): '(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 ff' (<type 'str'>)
```

```
Expectation (Extracted data): result = '(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 ff' (<type 'str'>)
```

A.1.9 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**.

Info	Creating testframe for ":testframe: for csp"

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

```
Result (CSP-Frame): ':testframe: for csp\n' (<type 'str'>)
```

```
Expectation (CSP-Frame): result = ':testframe: for csp\n' (<type 'str'>)
```

A.1.10 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**.

Info Creating testframe for ":testframe: for csp"

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

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

A.1.11 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**.

Info Processing testframe: ":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 <type 'list'>).

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

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

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

A.1.12 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**.

Info Processing wrong data (list)

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

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

Success Buffer still empty is correct (Content '' and Type is <type 'str'>).

```
Result (Buffer still empty): '' (<type 'str'>)
Expectation (Buffer still empty): result = '' (<type 'str'>)
```

Info Processing wrong data (int)

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

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

Success Buffer still empty is correct (Content '' and Type is <type 'str'>).

```
Result (Buffer still empty): '' (<type 'str'>)
Expectation (Buffer still empty): result = '' (<type 'str'>)
```

Info Processing wrong data (unicode)

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

```
Result (Wrong data exception): <type 'exceptions.ValueError'> (<type 'type'>)
```

```
Expectation (Wrong data exception): result = <type 'exceptions.ValueError'> (<type 'type'>)
```

Success	Buffer still empty is correct (Content '' and Type is <type 'str'>).

```
Result (Buffer still empty): '' (<type 'str'>)
```

```
Expectation (Buffer still empty): result = '' (<type 'str'>)
```

A.1.13 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**.

Info	Creating testframe for "testframe for stp"

Success	STP-Frame is correct (Content ':<testframe for stp:>' and Type is <type 'str'>).

```
Result (STP-Frame): ':<testframe for stp:>' (<type 'str'>)
```

```
Expectation (STP-Frame): result = ':<testframe for stp:>' (<type 'str'>)
```

A.1.14 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**.

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

Success	STP-Frame is correct (Content ':<testframe for :=<stp:=>:>' and Type is <type 'str'>).

```
Result (STP-Frame): ':<testframe for :=<stp:=>:>' (<type 'str'>)
```
```
Expectation (STP-Frame): result = ':<testframe for :=<stp:=>:>' (<type 'str'>)
```

A.1.15 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**.

Info	Processing testframe: ":<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 <type 'list'>).

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

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

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

A.1.16 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**.

Info Processing wrong data (list)

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

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

Success Buffer still empty is correct (Content " and Type is <type 'str'>).

```
Result (Buffer still empty): '' (<type 'str'>)
Expectation (Buffer still empty): result = '' (<type 'str'>)
```

Info Processing wrong data (int)

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

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

Success Buffer still empty is correct (Content " and Type is <type 'str'>).

```
Result (Buffer still empty): '' (<type 'str'>)
Expectation (Buffer still empty): result = '' (<type 'str'>)
```

Info Processing wrong data (unicode)

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

```
Result (Wrong data exception): <type 'exceptions.ValueError'> (<type 'type'>)
```

```
Expectation (Wrong data exception): result = <type 'exceptions.ValueError'> (<type 'type'>)
```

Success Buffer still empty is correct (Content '' and Type is <type 'str'>).

```
Result (Buffer still empty): '' (<type 'str'>)
```

```
Expectation (Buffer still empty): result = '' (<type 'str'>)
```

A.1.17 Frame processing - Start pattern and end pattern inside a message

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

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

Info Processing testframe: '':<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 ['testframe for :<stp:>'] and Type is <type 'list'>).

```
Result (Processed STP-Frame): [ 'testframe for :<stp:>' ] (<type 'list'>)
```

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

A.1.18 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**.

Info	Processing testframe: "_:<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 ['testframe for stp'] and Type is <type 'list'>).

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

A.1.19 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**.

Info	Processing data with an insufficient start pattern.

```
Sending ':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 <type 'list'>).

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

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

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

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

```
Sending '::' 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 <type 'list'>).

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

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

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

A.1.20 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**.

Info Processing data with an insufficient end pattern.

```
Sending ':<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 <type 'list'>).

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

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

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

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

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

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

```
Sending ':<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 <type 'list'>).

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

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

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

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

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

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

```
Sending ':<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 <type 'list'>).

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

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

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

A.1.21 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**.

Info Corrupting stp state and processing data.

```
Sending ':<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 ':<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 <type 'list'>).

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

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

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

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

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

B Trace for testrun with python 3.6.9 (final)

B.1 Tests with status Info (21)

B.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**.

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 '25.3' and Type is <class 'str'>).

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

B.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**.

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'>)
```

B.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**.

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'>)
```

B.1.4 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**.

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
```

B.1.5 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**.

Info	Checking test pattern with length 4.

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

B.1.6 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**.

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'>)
```

B.1.7 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**.

Info	Compressing Streams result in diffsame 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 ff ff

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

Info	Extracting stream: (26): 1f 8b 08 00 70 5b 38 5e 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 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 ff ff ff ff ff ff
↪  ff ff ff ff ff ff ff ff ff (<class 'bytes'>)
Expectation (Extracted data): result = (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 ff (<class 'bytes'>)
```

B.1.8 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**.

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 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 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 ff
↪  ff ff ff ff ff ff ff ff ff ff ff ff ff ff' (<class 'str'>)
```

B.1.9 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**.

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'>)
```

B.1.10 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**.

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'>)
```

B.1.11 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**.

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'>)
```

B.1.12 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**.

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'>)
```

B.1.13 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**.

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'>)
```

B.1.14 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**.

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'>)
```

B.1.15 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**.

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'>)
```

B.1.16 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**.

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'>)
```

B.1.17 Frame processing - Start pattern and end pattern inside a message

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

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'>)
```

B.1.18 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**.

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'>)
```

B.1.19 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**.

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'>)
```

B.1.20 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**.

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'>)
```

B.1.21 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**.

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'>)
```

C Test-Coverage

C.1 stringtools

The line coverage for stringtools was 100.0%
The branch coverage for stringtools was 97.7%

C.1.1 stringtools.__init__.py

The line coverage for stringtools.__init__.py was 100.0%
The branch coverage for stringtools.__init__.py was 97.7%

```
#!/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
if sys.version_info < (3, 0):
    from cStringIO import StringIO

logger_name = 'STRINGTOOLS'
logger = logging.getLogger(logger_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__ = (2, 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: ' ',
        -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.):
        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

    **Example:**

    .. literalinclude:: ../examples/gzip_compress.py

    Will result to the following output:

    .. literalinclude:: ../examples/gzip_compress.log
    """
    rv = None
    t = time.time()
    if sys.version_info >= (3, 0):
        rv = gzip.compress(s, compresslevel)
    else:
        buf = StringIO()
        f = gzip.GzipFile(mode='wb', compresslevel=compresslevel, fileobj=buf)
        try:
            f.write(s)
        finally:
            f.close()
            rv = buf.getvalue()
            buf.close()
    if rv is not None:
        logger.debug('GZIP: Finished to compress a string (compression_rate=%.3f, consumed_time
=%.1fs).', len(rv) / float(len(s)), time.time() - t)
    return rv


def gzip_extract(s):
    """
    Method to extract data from a compress stream of bytes.

    :param str s: The compressed bytestream (string) to be extracted
    :return: The extracted data
    :rtype: str

    **Example:**

    .. literalinclude:: ../examples/gzip_extract.py

    Will result to the following output:

    .. literalinclude:: ../examples/gzip_extract.log
    """
    t = time.time()
    rv = None
    if sys.version_info >= (3, 0):
        rv = gzip.decompress(s)
    else:
        inbuffer = StringIO(s)
        f = gzip.GzipFile(mode='rb', fileobj=inbuffer)
        try:
            rv = f.read()
        finally:
            f.close()
            inbuffer.close()
    if rv is not None:
        logger.debug('GZIP: Finished to extract a string (compression_rate=%.3f, consumed_time=%.1fs).', len(s) / float(len(rv)), time.time() - t)
    return rv


def hexlify(s):
    """Method to hexlify a string.

    :param str s: A string including the bytes to be hexlified.
    :returns: The hexlified string
    :rtype: str

    **Example:**

    .. literalinclude:: ../examples/hexlify.py

    Will result to the following output:

    .. literalinclude:: ../examples/hexlify.log
    """
    rv = '(%d):' % len(s)
    for byte in s:
        if sys.version_info >= (3, 0):
            rv += ' %02x' % byte
        else:
            rv += ' %02x' % ord(byte)
    return rv


def linefeed_filter(s):
    """Method to change linefeed and carriage return to '\\\\n' and '\\\\r'

    :param str s: A string including carriage return and/ or linefeed.
    :returns: A string with converted carriage return and/ or linefeed.
    :rtype: str
    """
    if sys.version_info >= (3, 0):
        return s.replace(b'\r', b'\\r').replace(b'\n', b'\\n')
    else:
        return s.replace('\r', '\\r').replace('\n', '\\n')
```

C.1.2 stringtools.csp.py

The line coverage for stringtools.csp.py was 100.0%
The branch coverage for stringtools.csp.py was 97.7%

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
"""
csp (Carriage-Return seperation protocol)
=========================================

**Author:**

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

```

```
**Description:**

    This module is a submodule of :mod:`stringtools` and creates an frame to transmit and receive
     messages via an serial interface.

**Submodules:**

* :class:`stringtools.csp.csp`
* :func:`stringtools.csp.build_frame`
"""

import stringtools

import logging
import sys

logger_name = 'STRINGTOOLS'
logger = logging.getLogger(logger_name)

DATA_SEPERATOR = b'\n'


class csp(object):
    """This class extracts messages from an "csp-stream".

    **Example:**

    .. literalinclude:: ../examples/csp.csp.py

    Will result to the following output:

    .. literalinclude:: ../examples/csp.csp.log
    """
    LOG_PREFIX = 'CSP:'

    def __init__(self, seperator=DATA_SEPERATOR):
        self.__buffer__ = b''
        self.__seperator__ = seperator

    def process(self, data):
        """
        This processes a byte out of a "stp-stream".

        :param bytes data: A byte stream
        :returns: A list of the extracted message(s)
        :rtype: list
        """
        if sys.version_info < (3, 0):
            if type(data) is unicode:
                raise TypeError
        #
        rv = (self.__buffer__ + data).split(self.__seperator__)
        self.__buffer__ = rv.pop()
        if len(self.__buffer__) != 0:
            logger.debug('%s Leaving data in buffer (to be processed next time): %s', self.
    LOG_PREFIX, stringtools.hexlify(self.__buffer__))
        for msg in rv:
            logger.info('%s message identified - %s', self.LOG_PREFIX, stringtools.hexlify(msg))
        return rv


def build_frame(msg, seperator=DATA_SEPERATOR):
    """This Method builds an "csp-frame" to be transfered via a stream.

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

    **Example:**

    .. literalinclude:: ../examples/csp.build_frame.py

    Will result to the following output:

    .. literalinclude:: ../examples/csp.build_frame.log
    """
    if seperator in msg:
        raise ValueError
    else:
        return msg + seperator
```

C.1.3 stringtools.stp.py

The line coverage for `stringtools.stp.py` was 100.0%
The branch coverage for `stringtools.stp.py` was 97.7%

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
"""
stp (Serial transfer protocol)
==============================

**Author:**

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

**Description:**

    This module is a submodule of :mod:`stringtools` and creates an serial frame to transmit and
    receive messages via an serial interface.

**Submodules:**

* :class:`stringtools.stp.stp`
* :func:`stringtools.stp.build_frame`
"""

import stringtools

import logging
import sys

logger_name = 'STRINGTOOLS'
logger = logging.getLogger(logger_name)

DATA_SYNC = b'\x3a'
"""The data sync byte"""
DATA_CLEAR_BUFFER = b'\x3c'
"""The clear buffer byte ('\\\\x3a\\\\x3c' -> start of message)"""
DATA_VALID_MSG = b'\x3e'
"""The valid message byte ('\\\\x3a\\\\x3e' -> end of message)"""
```

```
DATA_STORE_SYNC_VALUE = b'\x3d'
"""The store sync value byte ('\\\\x3a\\\\x3d' -> '\\\\x3a' inside a message)"""

STP_STATE_IDLE = 0x00
"""Idle state definition (default)"""
STP_STATE_ESCAPE_1 = 0x01
"""Escape 1 state definition ('\\\\x3a\\\\x3c' found)"""
STP_STATE_ESCAPE_2 = 0x02
"""Escape 2 state definition ('\\\\x3a' found inside a message)"""
STP_STATE_STORE_DATA = 0x03
"""Store data state definition (start of message found; data will be stored)"""


class stp(object):
    """This class extracts messages from an "stp-stream".

    **Example:**

    .. literalinclude:: ../examples/stp.stp.py

    Will result to the following output:

    .. literalinclude:: ../examples/stp.stp.log
    """
    LOG_PREFIX = 'STP:'

    def __init__(self):
        self.state = STP_STATE_IDLE
        self.__buffer__ = b''
        self.__clear_buffer__()

    def __clear_buffer__(self):
        if len(self.__buffer__) > 0:
            logger.warning('%s Chunking "%s" from buffer', self.LOG_PREFIX, stringtools.hexlify(
    self.__buffer__))
        self.__buffer__ = b''

    def process(self, data):
        """
        This processes a byte out of a "stp-stream".

        :param bytes data: A byte stream
        :returns: The extracted message or None, if no message is identified yet
        :rtype: str
        """
        if type(data) is list:
            raise TypeError
        if sys.version_info <= (3, 0):
            if type(data) is unicode:
                raise TypeError
        #
        rv = []
        #
        while len(data) > 0:
            if sys.version_info >= (3, 0):
                b = bytes([data[0]])
            else:
                b = data[0]
            data = data[1:]
            #
            if self.state == STP_STATE_IDLE:
                if b == DATA_SYNC:
                    self.state = STP_STATE_ESCAPE_1
                    logger.debug('%s data sync (%02x) received => changing state STP_STATE_IDLE -> STP_STATE_ESCAPE_1', self.LOG_PREFIX, ord(b))
                else:
                    logger.warning('%s no data sync (%02x) received => ignoring byte', self.LOG_PREFIX, ord(b))
            elif self.state == STP_STATE_ESCAPE_1:
                if b == DATA_CLEAR_BUFFER:
                    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))
                    self.state = STP_STATE_STORE_DATA
                    self.__clear_buffer__()
                elif b != DATA_SYNC:
                    self.state = STP_STATE_IDLE
                    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))
                else:
                    logger.warning('%s 2nd data sync (%02x) received => keep state', self.LOG_PREFIX, ord(b))
            elif self.state == STP_STATE_STORE_DATA:
                if b == DATA_SYNC:
                    self.state = STP_STATE_ESCAPE_2
                    logger.debug('%s data sync (%02x) received => changing state STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2', self.LOG_PREFIX, ord(b))
                else:
                    self.__buffer__ += b
            elif self.state == STP_STATE_ESCAPE_2:
                if b == DATA_CLEAR_BUFFER:
                    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))
                    self.state = STP_STATE_STORE_DATA
                    self.__clear_buffer__()
                elif b == DATA_VALID_MSG:
                    self.state = STP_STATE_IDLE
                    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__()
```

```python
                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:: ../examples/stp.build_frame.py

    Will result to the following output:

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

    for byte in data:
        if sys.version_info >= (3, 0):
            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
```