

Unittest for stringtools

January 14, 2021

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 2.7.18 (final)	4
2.2	Test-Statistic for testrun with python 3.8.5 (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 2.7.18 (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	35
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	38
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 3.8.5 (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	51
B.1.10	Frame creation error	52
B.1.11	Frame processing	52
B.1.12	Frame processing - Input data type error	53
B.1.13	Frame creation	54
B.1.14	Frame creation - Start pattern and end pattern inside a message	55
B.1.15	Frame processing	55
B.1.16	Frame processing - Input data type error	56
B.1.17	Frame processing - Start pattern and end pattern inside a message	57
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	63
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	09b4d1c41b828c8d1ccb723fa1fd79a9
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	e82636461580a46d22b3bb33660ea78a
Testruns with	python 2.7.18 (final), python 3.8.5 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	Linux Mint 20.1 ulyssa
Hostname	ahorn
Kernel	5.4.0-60-generic (#67-Ubuntu SMP Tue Jan 5 18:31:36 UTC 2021)
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 2.7.18 (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	0.026s

2.2 Test-Statistic for testrun with python 3.8.5 (final)

Number of tests	21
Number of successfull tests	21
Number of possibly failed tests	0
Number of failed tests	0
Executionlevel	Full Test (all defined tests)
Time consumption	0.025s

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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (24)
Start-Time:	2021-01-14 00:35:29,175
Finished-Time:	2021-01-14 00:35:29,177
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 '25.3u' and Type is <type 'str'>).
Success	Physical representation for 0.1 is correct (Content '100m' and Type is <type 'str'>).
Success	Physical representation for 0 is correct (Content '0' 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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (24)
Start-Time:	2021-01-14 00:35:29,599
Finished-Time:	2021-01-14 00:35:29,601
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 '25.3u' and Type is <class 'str'>).
Success	Physical representation for 0.1 is correct (Content '100m' and Type is <class 'str'>).
Success	Physical representation for 0 is correct (Content '0' and Type is <class 'str'>).
Success	Physical representation for 1 is correct (Content '1' and Type is <class 'str'>).
Success	Physical representation for 1000 is correct (Content '1k' and Type is <class 'str'>).
Success	Physical representation for 1005001 is correct (Content '1.01M' and Type is <class 'str'>).
Success	Physical representation for 1004000000 is correct (Content '1G' and Type is <class 'str'>).
Success	Physical representation for 1003000000000 is correct (Content '1T' and Type is <class 'str'>).
Success	Physical representation for 10000000000000000 is correct (Content '10P' and Type is <class 'str'>).
Success	Physical representation for 17.17 is correct (Content '17.17' and Type is <class 'str'>).
Success	Physical representation for 117000 is correct (Content '117k' and Type is <class 'str'>).
Success	Physical representation for 117.17 is correct (Content '117.2' and Type is <class 'str'>).

3.1.2 Time representation**Description**

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

Testresult

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

Testrun:	python 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (25)
Start-Time:	2021-01-14 00:35:29,177
Finished-Time:	2021-01-14 00:35:29,179
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 3.8.5 (final)
----------	----------------------

Caller: /user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (25)
 Start-Time: 2021-01-14 00:35:29,602
 Finished-Time: 2021-01-14 00:35:29,603
 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 2.7.18 (final)
 Caller: /user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (26)
 Start-Time: 2021-01-14 00:35:29,179
 Finished-Time: 2021-01-14 00:35:29,180
 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 3.8.5 (final)
 Caller: /user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (26)
 Start-Time: 2021-01-14 00:35:29,603
 Finished-Time: 2021-01-14 00:35:29,603
 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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (29)
Start-Time:	2021-01-14 00:35:29,180
Finished-Time:	2021-01-14 00:35:29,180
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (29)
Start-Time:	2021-01-14 00:35:29,604
Finished-Time:	2021-01-14 00:35:29,604
Time-Consumption	0.000s

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 brackets 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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (30)
Start-Time:	2021-01-14 00:35:29,181
Finished-Time:	2021-01-14 00:35:29,181
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (30)
Start-Time:	2021-01-14 00:35:29,604
Finished-Time:	2021-01-14 00:35:29,604
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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (31)
Start-Time:	2021-01-14 00:35:29,181
Finished-Time:	2021-01-14 00:35:29,181
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (31)
Start-Time:	2021-01-14 00:35:29,604
Finished-Time:	2021-01-14 00:35:29,604
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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (34)
Start-Time:	2021-01-14 00:35:29,181
Finished-Time:	2021-01-14 00:35:29,182
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 00 ff ff ff ff ff ff ff ff ff ff ff ff ff
Info	Extracting stream: (26): 1f 8b 08 00 c1 83 ff 5f 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 00 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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/__init__.py (34)
Start-Time:	2021-01-14 00:35:29,605
Finished-Time:	2021-01-14 00:35:29,605
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 00 ff ff ff ff ff ff ff ff ff ff ff ff ff
Info	Extracting stream: (26): 1f 8b 08 00 c1 83 ff 5f 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 00 ff ff ff ff ff ff ff ff ff ff ff ff ff and Type is <class 'bytes'>).

3.4.2 Extract

Description

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

Reason for the implementation

Speed up transfer with low transfer rate.

Fitcriterion

Extracted Stream is equal to the original compressed data.

Testresult

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

Testrun:	python 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (35)
Start-Time:	2021-01-14 00:35:29,182
Finished-Time:	2021-01-14 00:35:29,183
Time-Consumption	0.000s
Testsummary:	
Info	Extracting stream: (26): 1f 8b 08 00 34 e0 04 5d 02 ff 63 60 40 01 ff 51 01 00 2d 8a 7d de 1e 00 00 00
Success	Extracted data is correct (Content '(30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (35)
Start-Time:	2021-01-14 00:35:29,606
Finished-Time:	2021-01-14 00:35:29,606
Time-Consumption	0.000s
Testsummary:	
Info	Extracting stream: (26): 1f 8b 08 00 34 e0 04 5d 02 ff 63 60 40 01 ff 51 01 00 2d 8a 7d de 1e 00 00 00
Success	Extracted data is correct (Content '(30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff' and Type is <class 'str'>).

3.5 Carriagereturn Seperation Protocol (CSP)

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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (39)
Start-Time:	2021-01-14 00:35:29,183
Finished-Time:	2021-01-14 00:35:29,183
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (39)
Start-Time:	2021-01-14 00:35:29,606
Finished-Time:	2021-01-14 00:35:29,606
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

ValueError 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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (40)
Start-Time:	2021-01-14 00:35:29,183
Finished-Time:	2021-01-14 00:35:29,183
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unititest/stringtools/unititest/src/tests/_init_.py (40)
Start-Time:	2021-01-14 00:35:29,606
Finished-Time:	2021-01-14 00:35:29,607
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 snippets 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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unititest/stringtools/unititest/src/tests/_init_.py (41)
Start-Time:	2021-01-14 00:35:29,183
Finished-Time:	2021-01-14 00:35:29,184
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unititest/stringtools/unititest/src/tests/_init_.py (41)
Start-Time:	2021-01-14 00:35:29,607
Finished-Time:	2021-01-14 00:35:29,607

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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (42)
Start-Time:	2021-01-14 00:35:29,184
Finished-Time:	2021-01-14 00:35:29,185
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 3.8.5 (final)
----------	----------------------

Caller: /user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/...init....py (42)
 Start-Time: 2021-01-14 00:35:29,607
 Finished-Time: 2021-01-14 00:35:29,609
 Time-Consumption 0.002s

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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/...init....py (46)
Start-Time:	2021-01-14 00:35:29,185
Finished-Time:	2021-01-14 00:35:29,185
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (46)
Start-Time:	2021-01-14 00:35:29,609
Finished-Time:	2021-01-14 00:35:29,610
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 existence 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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (47)
Start-Time:	2021-01-14 00:35:29,186
Finished-Time:	2021-01-14 00:35:29,186
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (47)
Start-Time:	2021-01-14 00:35:29,610
Finished-Time:	2021-01-14 00:35:29,610
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 snippets 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.

Fitterion

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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (48)
Start-Time:	2021-01-14 00:35:29,186
Finished-Time:	2021-01-14 00:35:29,187
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (48)
Start-Time:	2021-01-14 00:35:29,611
Finished-Time:	2021-01-14 00:35:29,613
Time-Consumption	0.002s

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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (49)
Start-Time:	2021-01-14 00:35:29,187
Finished-Time:	2021-01-14 00:35:29,190
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (49)
Start-Time:	2021-01-14 00:35:29,614
Finished-Time:	2021-01-14 00:35:29,616
Time-Consumption	0.002s
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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (50)
Start-Time:	2021-01-14 00:35:29,190
Finished-Time:	2021-01-14 00:35:29,193
Time-Consumption	0.003s

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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (50)
Start-Time:	2021-01-14 00:35:29,616
Finished-Time:	2021-01-14 00:35:29,618
Time-Consumption	0.002s

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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (51)
Start-Time:	2021-01-14 00:35:29,193
Finished-Time:	2021-01-14 00:35:29,195
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (51)
Start-Time:	2021-01-14 00:35:29,618
Finished-Time:	2021-01-14 00:35:29,619
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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (52)
Start-Time:	2021-01-14 00:35:29,195
Finished-Time:	2021-01-14 00:35:29,198
Time-Consumption	0.003s
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (52)
Start-Time:	2021-01-14 00:35:29,619
Finished-Time:	2021-01-14 00:35:29,621
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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init....py (53)
Start-Time:	2021-01-14 00:35:29,198
Finished-Time:	2021-01-14 00:35:29,201
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unititest/stringtools/unititest/src/tests/_init....py (53)
Start-Time:	2021-01-14 00:35:29,621
Finished-Time:	2021-01-14 00:35:29,625
Time-Consumption	0.004s

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 2.7.18 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (54)
Start-Time:	2021-01-14 00:35:29,201
Finished-Time:	2021-01-14 00:35:29,204
Time-Consumption	0.003s
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 3.8.5 (final)
Caller:	/user_data/data/dirk/prj/unittest/stringtools/unittest/src/tests/_init_.py (54)
Start-Time:	2021-01-14 00:35:29,625
Finished-Time:	2021-01-14 00:35:29,627
Time-Consumption	0.002s
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 2.7.18 (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 '25.3u' and Type is <type 'str'>).

Result (Physical representation for 2.53e-05): '25.3u' (<type 'str'>)

Expectation (Physical representation for 2.53e-05): result = '25.3u' (<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 0 is correct (Content '0' and Type is <type 'str'>).

Result (Physical representation for 0): '0' (<type 'str'>)

Expectation (Physical representation for 0): result = '0' (<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 brackets 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 different streams with the same input stream. Therefore the test will compare the decompressed data.

[illegible]

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

Info Extracting stream: (26): 1f 8b 08 00 c1 83 ff 5f 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).
```

[illegible]

```
Result (Extracted data): (30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff ff ff
↳ ff ff ff ff ff ff ff ff ff ff (<type 'str'>)
```

```
Expectation (Extracted data): result = (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 (<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 00 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 00 ff ff ff 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 00 ff
↪ 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

ValueError 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 snippets 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 existence 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 snippets 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'>)
↪ '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'>)
↪]] (<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'>)
↪ '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 3.8.5 (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 '25.3u' and Type is <class 'str'>).

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

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

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

Result (Physical representation for 0.1): '100m' (<class 'str'>)

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

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

Result (Physical representation for 0): '0' (<class 'str'>)

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

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

Result (Physical representation for 1): '1' (<class 'str'>)

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

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

Result (Physical representation for 1000): '1k' (<class 'str'>)

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

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

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

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

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

Result (Physical representation for 1004000000): '1G' (<class 'str'>)

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

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

Result (Physical representation for 1003000000000): '1T' (<class 'str'>)

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

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

Result (Physical representation for 10000000000000000): '10P' (<class 'str'>)

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

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

Result (Physical representation for 17.17): '17.17' (<class 'str'>)

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

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

Result (Physical representation for 117000): '117k' (<class 'str'>)

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

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

Result (Physical representation for 117.17): '117.2' (<class 'str'>)

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

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 differnt streams with the same input stream. Therefore the test will compare the decompressed data.

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

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

Info Extracting stream: (26): 1f 8b 08 00 c1 83 ff 5f 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).
```

[illegible]

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

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

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

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

```
Expectation (Extracted data): result = '(30): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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

ValueError 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 snippets 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 existence 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 snippets 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'>)
↪ []] (<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'>)
↪ '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'>)
↪ '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'>)
↪]] (<class 'list'>)

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

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #
4 """
5 stringtools (Stringtools)
6 =====
7
8 **Author:**
9
10 * Dirk Alders <sudo-dirk@mount-mockery.de>
11
12 **Description:**
13
14     This Module supports functionality around string operations.
15
16 **Submodules:**
17
18 * :mod:`stringtools.csp`
19 * :mod:`stringtools.stp`
20 * :func:`gzip_compress`
21 * :func:`gzip_extract`
22 * :func:`hexlify`
23
24 **Unittest:**
25

```


Unittest for stringtools

```
26         See also the :download:`unittest <stringtools/_testresults_/unittest.pdf>` documentation.
27
28     **Module Documentation:**
29
30     """
31
32     from stringtools import stp
33     from stringtools import csp
34     __DEPENDENCIES__ = []
35
36     import fractions
37     import gzip
38     import logging
39     import time
40     import sys
41     if sys.version_info < (3, 0):
42         from cStringIO import StringIO
43
44     try:
45         from config import APP_NAME as ROOT_LOGGER_NAME
46     except ImportError:
47         ROOT_LOGGER_NAME = 'root'
48     logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)
49
50     __DESCRIPTION__ = """The Module {\\tt %s} is designed to support functionality for strings (e.g.
51         transfer strings via a bytestream, compressing, extracting, ...).
52     For more information read the sphinx documentation.""" % __name__.replace('-', '\\-')
53     """The Module Description"""
54     __INTERPRETER__ = (2, 3)
55     """The Tested Interpreter—Versions"""
56
57     __all__ = ['gzip_compress',
58               'gzip_extract',
59               'hexlify',
60               'csp',
61               'stp']
62
63     def physical_value_repr(value, unit=''):
64         prefix = {
65             -4: 'p',
66             -3: 'n',
67             -2: 'u',
68             -1: 'm',
69             0: '',
70             1: 'k',
71             2: 'M',
72             3: 'G',
73             4: 'T',
74             5: 'P',
75         }
76         u = 0
77         while u > -4 and u < 5 and (value >= 1000. or value < 1.) and value != 0:
78             if value >= 1:
79                 u += 1
80                 value /= 1000.
81             else:
82                 u -= 1
83                 value *= 1000.
84         if u == 0:
85             ext = ''
86         else:
```

```

87     ext = prefix[u]
88     #
89     if value < 100.:
90         value = '%.2f' % (value)
91     else:
92         value = '%.1f' % (value)
93     while value.find('.') > -1 and (value.endswith('0') or value.endswith('.')):
94         value = value[:-1]
95     return value + ext + unit
96
97
98 def time_repr(seconds):
99     days = seconds / (24 * 60 * 60)
100     seconds = seconds % (24 * 60 * 60)
101     if seconds >= 60 * 60:
102         rv = time.strftime('%H:%M:%S', time.gmtime(seconds))
103     else:
104         rv = time.strftime('%M:%S', time.gmtime(seconds))
105     if days >= 1:
106         rv = '%dD %s' % (days, rv)
107     if rv.endswith(' 00:00'):
108         rv = rv[:-6]
109     return rv
110
111
112 def frac_repr(value):
113     f = fractions.Fraction(value).limit_denominator()
114     return '%s/%s' % (f.numerator, f.denominator)
115
116
117 def gzip_compress(s, compresslevel=9):
118     """
119     Method to compress a stream of bytes.
120
121     :param str s: The bytestream (string) to be compressed
122     :param int compresslevel: An optional compression level (default is 9)
123     :return: The compressed bytestream
124     :rtype: str
125
126     **Example:**
127
128     .. literalinclude:: stringtools/_examples_/gzip_compress.py
129
130     Will result to the following output:
131
132     .. literalinclude:: stringtools/_examples_/gzip_compress.log
133     """
134     rv = None
135     t = time.time()
136     if sys.version_info >= (3, 0):
137         rv = gzip.compress(s, compresslevel)
138     else:
139         buf = StringIO()
140         f = gzip.GzipFile(mode='wb', compresslevel=compresslevel, fileobj=buf)
141         try:
142             f.write(s)
143         finally:
144             f.close()
145             rv = buf.getvalue()
146             buf.close()
147     if rv is not None:

```

```

148     logger.debug('GZIP: Finished to compress a string (compression_rate=%.3f, consumed_time
149     =%.1fs).', len(rv) / float(len(s)), time.time() - t)
150     return rv
151
152 def gzip_extract(s):
153     """
154     Method to extract data from a compress stream of bytes.
155
156     :param str s: The compressed bytestream (string) to be extracted
157     :return: The extracted data
158     :rtype: str
159
160     **Example:**
161
162     .. literalinclude:: stringtools/_examples_/gzip_extract.py
163
164     Will result to the following output:
165
166     .. literalinclude:: stringtools/_examples_/gzip_extract.log
167     """
168     t = time.time()
169     rv = None
170     if sys.version_info >= (3, 0):
171         rv = gzip.decompress(s)
172     else:
173         inbuffer = StringIO(s)
174         f = gzip.GzipFile(mode='rb', fileobj=inbuffer)
175         try:
176             rv = f.read()
177         finally:
178             f.close()
179             inbuffer.close()
180     if rv is not None:
181         logger.debug('GZIP: Finished to extract a string (compression_rate=%.3f, consumed_time
182         =%.1fs).', len(s) / float(len(rv)), time.time() - t)
183         return rv
184
185 def hexlify(s):
186     """Method to hexlify a string.
187
188     :param str s: A string including the bytes to be hexlified.
189     :returns: The hexlified string
190     :rtype: str
191
192     **Example:**
193
194     .. literalinclude:: stringtools/_examples_/hexlify.py
195
196     Will result to the following output:
197
198     .. literalinclude:: stringtools/_examples_/hexlify.log
199     """
200     rv = '(%d):' % len(s)
201     for byte in s:
202         if sys.version_info >= (3, 0):
203             rv += ' %02x' % byte
204         else:
205             rv += ' %02x' % ord(byte)
206     return rv

```

```

207
208
209 def linefeed_filter(s):
210     """Method to change linefeed and carriage return to '\\\\n' and '\\\\r'
211
212     :param str s: A string including carriage return and/ or linefeed.
213     :returns: A string with converted carriage return and/ or linefeed.
214     :rtype: str
215     """
216     if sys.version_info >= (3, 0):
217         return s.replace(b'\r', b'\\r').replace(b'\n', b'\\n')
218     else:
219         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%

```

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

```

```

37 """ This class extracts messages from an "csp-stream".
38
39 **Example:**
40
41 .. literalinclude:: stringtools/_examples_/csp.csp.py
42
43 Will result to the following output:
44
45 .. literalinclude:: stringtools/_examples_/csp.csp.log
46 """
47 LOG_PREFIX = 'CSP:'
48
49 def __init__(self, seperator=DATA_SEPERATOR):
50     self.__buffer__ = b''
51     self.__seperator__ = seperator
52
53 def process(self, data):
54     """
55     This processes a byte out of a "stp-stream".
56
57     :param bytes data: A byte stream
58     :returns: A list of the extracted message(s)
59     :rtype: list
60     """
61     if sys.version_info < (3, 0):
62         if type(data) is unicode:
63             raise TypeError
64
65     #
66     rv = (self.__buffer__ + data).split(self.__seperator__)
67     self.__buffer__ = rv.pop()
68     if len(self.__buffer__) != 0:
69         logger.debug('%s Leaving data in buffer (to be processed next time): %s', self.
70 LOG_PREFIX, stringtools.hexlify(self.__buffer__))
71     for msg in rv:
72         logger.info('%s message identified - %s', self.LOG_PREFIX, stringtools.hexlify(msg))
73     return rv
74
75 def build_frame(msg, seperator=DATA_SEPERATOR):
76     """ This Method builds an "csp-frame" to be transfered via a stream.
77
78     :param str data: A String (Bytes) to be framed
79     :returns: The "csp-framed" message to be sent
80     :rtype: str
81
82     **Example:**
83
84     .. literalinclude:: stringtools/_examples_/csp.build_frame.py
85
86 Will result to the following output:
87
88 .. literalinclude:: stringtools/_examples_/csp.build_frame.log
89 """
90 if seperator in msg:
91     raise ValueError
92 else:
93     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%

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  #
4  """
5  stringtools.stp (Serial transfer protocol)
6  =====
7
8  **Author:**
9
10 * Dirk Alders <sudo-dirk@mount-mockery.de>
11
12 **Description:**
13
14     This module is a submodule of :mod:`stringtools` and creates an serial frame to transmit and
15     receive messages via an serial interface.
16
17 **Submodules:**
18
19 * :class:`stringtools.stp.stp`
20 * :func:`stringtools.stp.build_frame`
21 """
22 import stringtools
23
24 import logging
25 import sys
26
27 try:
28     from config import APP_NAME as ROOT_LOGGER_NAME
29 except ImportError:
30     ROOT_LOGGER_NAME = 'root'
31 logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)
32
33 DATA_SYNC = b'\x3a'
34 """The data sync byte"""
35 DATA_CLEAR_BUFFER = b'\x3c'
36 """The clear buffer byte ('\x3a\x3c' -> start of message)"""
37 DATA_VALID_MSG = b'\x3e'
38 """The valid message byte ('\x3a\x3e' -> end of message)"""
39 DATA_STORE_SYNC.VALUE = b'\x3d'
40 """The store sync value byte ('\x3a\x3d' -> '\x3a' inside a message)"""
41
42 STP_STATE_IDLE = 0x00
43 """Idle state definition (default)"""
44 STP_STATE_ESCAPE_1 = 0x01
45 """Escape 1 state definition ('\x3a\x3c' found)"""
46 STP_STATE_ESCAPE_2 = 0x02
47 """Escape 2 state definition ('\x3a' found inside a message)"""
48 STP_STATE_STORE_DATA = 0x03
49 """Store data state definition (start of message found; data will be stored)"""
50
51
52 class stp(object):

```

```

53     """ This class extracts messages from an "stp-stream".
54
55     **Example:**
56
57     .. literalinclude:: stringtools/_examples_/stp.stp.py
58
59     Will result to the following output:
60
61     .. literalinclude:: stringtools/_examples_/stp.stp.log
62     """
63     LOG_PREFIX = 'STP:'
64
65     def __init__(self):
66         self.state = STP_STATE.IDLE
67         self.__buffer__ = b''
68         self.__clear_buffer__()
69
70     def __clear_buffer__(self):
71         if len(self.__buffer__) > 0:
72             logger.warning('%s Chunking "%s" from buffer', self.LOG_PREFIX, stringtools.hexlify(
73                 self.__buffer__))
74             self.__buffer__ = b''
75
76     def process(self, data):
77         """
78         This processes a byte out of a "stp-stream".
79
80         :param bytes data: A byte stream
81         :returns: The extracted message or None, if no message is identified yet
82         :rtype: str
83         """
84         if type(data) is list:
85             raise TypeError
86         if sys.version_info <= (3, 0):
87             if type(data) is unicode:
88                 raise TypeError
89
90         #
91         rv = []
92         #
93         while len(data) > 0:
94             if sys.version_info >= (3, 0):
95                 b = bytes([data[0]])
96             else:
97                 b = data[0]
98                 data = data[1:]
99             #
100             if self.state == STP_STATE.IDLE:
101                 if b == DATA_SYNC:
102                     self.state = STP_STATE.ESCAPE_1
103                     logger.debug('%s data sync (%02x) received => changing state STP_STATE.IDLE
104                     -> STP_STATE.ESCAPE_1', self.LOG_PREFIX, ord(b))
105                 else:
106                     logger.warning('%s no data sync (%02x) received => ignoring byte', self.
107                     LOG_PREFIX, ord(b))
108                 elif self.state == STP_STATE.ESCAPE_1:
109                     if b == DATA_CLEAR_BUFFER:
110                         logger.debug('%s start pattern (%02x %02x) received => changing state
111                         STP_STATE.ESCAPE_1 -> STP_STATE.STORE_DATA', self.LOG_PREFIX, ord(DATA_SYNC), ord(b))
112                         self.state = STP_STATE.STORE_DATA
113                         self.__clear_buffer__()
114                     elif b != DATA_SYNC:
115                         self.state = STP_STATE.IDLE
116                         logger.warning('%s no start pattern (%02x %02x) received => changing state
117                         STP_STATE.ESCAPE_1 -> STP_STATE.IDLE', self.LOG_PREFIX, ord(DATA_SYNC), ord(b))

```

```

112         else:
113             logger.warning('%s 2nd data sync (%02x) received => keep state', self.
LOG_PREFIX, ord(b))
114         elif self.state == STP_STATE_STORE_DATA:
115             if b == DATA_SYNC:
116                 self.state = STP_STATE_ESCAPE_2
117                 logger.debug('%s data sync (%02x) received => changing state
STP_STATE_STORE_DATA -> STP_STATE_ESCAPE_2', self.LOG_PREFIX, ord(b))
118             else:
119                 self.__buffer__ += b
120         elif self.state == STP_STATE_ESCAPE_2:
121             if b == DATA_CLEAR_BUFFER:
122                 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))
123                 self.state = STP_STATE_STORE_DATA
124                 self.__clear_buffer__()
125             elif b == DATA_VALID_MSG:
126                 self.state = STP_STATE_IDLE
127                 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)
)
128                 rv.append(self.__buffer__)
129                 self.__buffer__ = b''
130             elif b == DATA_STORE_SYNC_VALUE:
131                 self.state = STP_STATE_STORE_DATA
132                 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))
133                 self.__buffer__ += DATA_SYNC
134             elif b == DATA_SYNC:
135                 self.state = STP_STATE_ESCAPE_1
136                 logger.warning('%s second data sync (%02x) received => changing state
STP_STATE_ESCAPE_2 -> STP_STATE_ESCAPE_1', self.LOG_PREFIX, ord(b))
137                 self.__clear_buffer__()
138         else:
139             self.state = STP_STATE_IDLE
140             logger.warning('%s data (%02x) received => changing state STP_STATE_ESCAPE_2
-> STP_STATE_IDLE', self.LOG_PREFIX, ord(b))
141             self.__clear_buffer__()
142         else:
143             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))
144             self.state = STP_STATE_IDLE
145             self.__clear_buffer__()
146             data = b + data
147         for msg in rv:
148             logger.info('%s message identified - %s', self.LOG_PREFIX, stringtools.hexlify(msg))
149         return rv
150
151
152 def build_frame(data):
153     """This Method builds an "stp-frame" to be transfered via a stream.
154
155     :param str data: A String (Bytes) to be framed
156     :returns: The "stp-framed" message to be sent
157     :rtype: str
158
159     **Example:**
160
161     .. literalinclude:: stringtools/_examples_/stp.build_frame.py
162
163     Will result to the following output:
164
165     .. literalinclude:: stringtools/_examples_/stp.build_frame.log
166     """

```



```
167     rv = DATA_SYNC + DATA_CLEAR_BUFFER
168
169     for byte in data:
170         if sys.version_info >= (3, 0):
171             byte = bytes([byte])
172         if byte == DATA_SYNC:
173             rv += DATA_SYNC + DATA_STORE_SYNC_VALUE
174         else:
175             rv += byte
176
177     rv += DATA_SYNC + DATA_VALID_MSG
178     return rv
```