Document Release Status	
Released	
Release Date	Version
4/23/2013	001

Versatile Binary Format Test Specification v3.0

Part must comply with material specification:
WSS-M99P9999-A1 to help safeguard health, safety and the environment.

Document Name
VERSATILE BINARY FORMAT TEST SPECIFICATION V3.0

Document Type
TEST SPECIFICATION

Owner Domain:Document Prefix

Document No	Revision	Volume No	Page No (In this doc.)
00.06.15.244	001	01	1 (11)

Error! Unknown

Revision history

Previous Version	Current Version	Version Description	Responsible	Date
N/A	001	1st Official Release	Vijaya Pinnamaneni	23-Apr-2013

Change log

Release	Section	Change Description
001	All	Changed from Volvo format to Ford Format
001	All	All the test cases related Block_Checkcum and Network identifier are deleted.

Contents

List of Tables

1 Introduction

1.1 Purpose/Scope

This procedure shall be used to certify the conformance with requirements as specified in the "Versatile Binary Format V2.4 Specification".

1.2 Use of this document

The tests within may be applied to any ECU implementation that has been designed in accordance to " Versatile Binary Format Specification 3.0 [5]", and "Software Download Specification [4]" as identified in 1.3 of this document. Individual test procedures of this document may be performed in various sequences, subsets, and repetitively, for the purposes of evaluation and development.

Name of this test specification identifies the version of VBF Format whereas Revision and Volume numbers represent official and draft releases respectively.

1.3 Applicable documents

The following documents are either referenced by this specification, or contain information that is relevant to this specification.

Table 1: Applicable Documents

Reference Num	Source	Title	Version or date	Document Number
[1]	ISO	Road vehicles – Unified diagnostic services (UDS) – Part 1: Specification and requirements [//TODO: Add latest version prior to release]	2011-08-23	ISO/FDIS 14229-1
[2]	ISO	Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services	2011-11-15	ISO 15765-2:2011
[3]	ISO	Road vehicles – Unified diagnostics services (UDS) – Part 2: Session layer services [//TODO: Add latest version prior to release]	2011-08-23	ISO/FDIS 14229-2
[4]	FMC	Software Download Specification	006, [//TODO: Add latest version prior to release]	00.06.15.002
[5]	FMC	Versatile Binary Format Specification 3.0	007, [//TODO: Add latest version prior to release]	00.06.15.004
[6]	FMC	Data Compression and Encryption Specification	Latest Available	00.06.15.005

Ford requirements documents, such as this one may be accessed on the Ford Intranet at the R&VT/EESE on-line documentation site (https://f1.ford.com/eRoom/EESE/NetCom).

Information for obtaining ISO documents may be found on the external Internet: “http:// www.iso.ch”.

1.4 Abbreviations/Acronyms

The following abbreviations are used throughout this specification:

CM:	**C**o**M**ments
EESE:	**E**lectrical/**E**lectronic **S**ystems **E**ngineering
GBL:	**G**ateway **B**oot**l**oader
PBL:	**P**rimary **B**oot**l**oader
SBL:	**S**econdary **B**oot**l**oader
VBF:	**V**ersatile **B**inary **F**ormat
WS:	**W**hite **S**pace

2 Test Provisions

2.1 Test Tool

The following test tool is recommended to perform all the test cases contained herein.

- Diagnostic Script Player (DSP)

2.2 Notation

2.2.1 Symbols

The following symbols are used throughout the tests denoting expected bytes or bits to send or receive:

G	=	unknown four bit frame type
L	=	unknown four or 12 bit length
HH	=	Data from ECU of unknown number of bytes
$	=	Hex
**	=	One byte of don't care data from ECU

2.2.2 Test Pass/Fail Criteria

The execution of each test within this test specification shall return a specific test result. The valid test results which may be returned are as follows:

- Test Failed
  - This test result shall be returned whenever a test execution completes and one or more steps resulted in an ERROR.
- Test Passed with Warnings
  - This test result shall be returned whenever a test execution completes and no steps resulted in an ERROR or TIMING_ERROR, but one or more steps resulted in a WARNING.
- Test Passed
  - This test result shall be returned whenever a test execution completes and no steps resulted in an ERROR, TIMING_ERROR or in a WARNING.

3 Test Cases for Versatile Binary Format

3.1 VBF File format Test

3.1.1 Data needed to run the test

3.1.1.1 Data

- VBF File

3.1.1.2 Assumptions/Special Handling

- Set ERROR, if there is any unexpected text in the file header which is not a white space or a comment
- Data block checksum shall be calculated using CRC16 polynomial with all the data in the data block (excluding start address, length, and checksum) as in reference [5].
- Data file checksum shall be calculated using CRC32 polynomial with all the data in the data section (including start address, length, and checksum for data blocks) as in reference [5].
- All identifiers and reserved words in the header are case sensitive
- White space characters or comments are always ignored in all the expressions with the following exception:
  - White space within quotes is not ignored
- Contents in grouping symbols [ ], brackets, are optional.

3.1.1.3 Notation

fileName = VBF File Name

3.1.2 Test Case

Test Case	Test Description	Issue (See Notes 1)
10	Verify that VBF file contains three sections:	
	1. vbf_version	1. No vbf_version present
	2. header	2. No header present
	3. data	3. No data section present
20	Verify the following for vbf_version identifier:	
	1. Very first character is 'v' and is not preceded by white space.	1. No vbf_version present
	2. Very first line is vbf_version[WS] = [WS]2.4[WS];[WS]	2. No or invalid vbf_version present
	3. No comments or non white space characters exist before, within, and after vbf_version expression	3. No vbf_version
30	Verify the following for header identifier:	
	1. Verify that next non white space character following vbf_version is 'h' followed by 'eader' and beginning braces as shown below: vbf_version=2.4[WS]; [WS]header[WS]{	1. No header present
	2. Verify that no comments or non white space characters exist between header identifier and beginning braces	2. No header present

	3. Verify that all the header information is contained in a set of braces { }	3. Invalid header
40	Verify the following for all identifiers:	
	1. All identifiers are contained within header section	1. Invalid text
	2. All identifiers match with one of the following reserved words: description, sw_part_number, sw_part_type, data_format_identifier, , ecu_address, frame_format, erase, omit, call, and file_checksum	2. Invalid field
	3. The file contains all mandatory identifiers: sw_part_number, sw_part_type, ecu_address, frame_format, and file_checksum	3. No (sw_part_number, sw_part_type, ecu_address, frame_format, or file_checksum) present
	4. Identifiers are not repeated	4. Duplicate (vbf_version, header, description, sw_part_number, sw_part_type, data_format_identifier, ecu_address, frame_format, erase, omit, call, or file_checksum)
50	Verify the following general format requirements for all the expressions in the header section:	
	1. [WS/CM]identifier[WS/CM] = [WS/CM]identifierValue[WS/CM];[WS/CM]	Invalid text or Invalid (description, sw_part_number, sw_part_type, data_format_identifier, ecu_address, frame_format, erase, omit, call, , or file_checksum)
	2. If an expression contains more than one value, verify that the optional field complies with [,[WS/CM]IdentifierValue[WS/CM]]	
	3. All the expressions are terminated by semi-colon (;)	
60	If description identifier exists, verify the following	If description field doesn't exist "No description" (See Notes 2) shall be set.
	1. The value of the description identifier is contained within one set of braces { }, whether there is a single row or multiple rows of description	1. Invalid description
	2. Each row is contained within quotes (" ") and is followed by a comma (,) if there is more than one row	2. Invalid description
	3. The last row ends with a quote (") and is followed by ending braces (})	3. Invalid description
	4. No quotes exist within the quotes	4. Invalid description
	5. Number of rows ≤ 16	5. Invalid description
	6. Number of characters in each row ≤ 80	6. Invalid description
70	Verify the following for sw_part_number identifier	
	1. sw_part_number identifier is assigned one or two values	1. Invalid sw_part_number
	2. If sw_part_number identifier is assigned two values, verify that part numbers are separated by comma (,)and are contained in braces {}	2. Invalid sw_part_number
	3. The sw_part_number identifier value is contained in quotes " "	3. Invalid sw_part_number
	4. sw_part_number identifier value is ≤ 24 characters of data	4. Invalid sw_part_number
	5. No white space characters or comments exist within quotes	5. Invalid sw_part_number

	6. Verify that VBF fileName is equal to sw_part_number + .VBF where sw_part_number shall be WERS	6. Invalid VBF file name
	7. If sw_part_number identifier contains only one value, verify that it is not contained in braces {}	7. Invalid sw_part_number
80	Verify the following for sw_part_type identifier	
	1. Software part type identifier value matches with one of the following reserved words: CARCFG, CUSTOM, DATA, EXE, GBL, SBL, SIGCFG, TEST	1. Invalid sw_part_type
	2. If sw_part_type = SBL or GBL, verify that erase identifier is not present in the VBF file header	2. Erase field not allowed
	3. sw_part_type is assigned only one value	3. Invalid sw_part_type
	4. sw_part_type identifier value is not contained in braces {}	4. Invalid sw_part_type
90	Verify the following for the data_format_identifier identifier (if present)	
	1. data_format_identifier = an invalid number, e.g. 0xFF (0x00 and 0x10 are valid data_format_identifiers)	1. Invalid data_format_identifier
100	Verify the following for frame_format identifier	
	1. frame_format identifier value matches with one of the following reserved words: CAN_STANDARD or CAN_EXTENDED	1. Invalid frame_format
	2. frame_format identifier value is not contained in braces {}	2. Invalid frame_format
110	Verify the following for ecu_address identifier	
	1. ecu_address identifier is assigned one or three values	1. Invalid ecu_address format
	2. If ecu_address identifier is assigned one value, verify that it is main node address,	2. Invalid main node address
	3. if ecu_address identifier is assigned more than one value, verify that they are main node address, sub network address, and sub node address separated by commas and contained in braces {}	3. Invalid main node, sub network, or sub node address
	4. If ecu_address identifier contains only one value, verify that it is not contained in braces {}	4. Invalid ecu_address format
	5. If ecu_address value is preceded by '0x', verify that the alphanumeric characters are from the string "0123456789ABCDEFG"	5. Invalid ecu_address
	6. If ecu_address value is preceded by '0b', verify that the digits followed are either '0' or '1'	6. Invalid ecu_address
	7. If ecu_address values are not preceded by '0x' or '0b', verify that the characters followed are from 0 - 9	7. Invalid ecu_address
120	If frame_format is CAN_STANDARD, and ecu_address identifier is assigned one value, verify that main node address is ≤ 0x7FF and ≥ 0	Invalid main node address
130	If frame_format is CAN_STANDARD, and ecu_address identifier is assigned three values, verify the following:	
	1. Main node address is 11 bits long, ≥ 0 and ≤ 0x7FF	1. Invalid main node address
	2. Sub network address is one byte long, ≥ 0 and ≤ 0xFF	2. Invalid sub network address
	3. Sub node address is one byte long, ≥ 0 and ≤ 0xFF	3. Invalid sub node address

140	If frame_format is CAN_EXTENDED, and ecu_address identifier is assigned one value, verify that main node address is one byte long, ≥ 0, and ≤ 0xFF	Invalid main node address
150	If frame_format is CAN_EXTENDED, and ecu_address identifier is assigned three values, verify the following	
	1. Main node address = 0x00	1. Invalid main node address
	2. Sub network address is one byte long, ≥ 0, and ≤ 0x07	2. Invalid sub network address
	3. Sub node address is one byte long, ≥ 0, and ≤ 0xFF	3. Invalid sub node address
160	Verify the following for erase identifier (if present)	
	1. erase identifier contains block information record with start address and length separated by comma and is contained in braces { }.	1. Invalid erase
	2. Address and length are four bytes long, ≥ 0, and ≤ 0xFFFFFFFF	2. Invalid erase
	3. erase identifier is assigned one or more block information record(s) separated by commas and is/are contained in braces {}	3. Invalid erase
	4. (Start address + Length) is not greater than 0xFFFFFFFF	4. Invalid erase
	5. Verify that erase identifier values are contained in two sets of braces	5. Invalid erase
	6. If erase values are preceded by '0x', verify that the alphanumeric characters are from the string "0123456789ABCDEFG"	6. Invalid erase
	7. If erase values are preceded by '0b', verify that the digits followed are either '0' or '1'	7. Invalid erase
	8. If erase values are not preceded by '0x' or '0b', verify that the characters followed are from 0 - 9	8. Invalid erase
170	Verify the following for omit identifier (if present)	
	1. omit identifier contains block information record with start address and length separated by comma and is contained in braces { }.	1. Invalid omit
	2. Address and length are four bytes long, ≥ 0, and ≤ 0xFFFFFFFF	2. Invalid omit
	3. omit identifier is assigned one or more block information record(s) separated by commas and is/are contained in braces {}	3. Invalid omit
	4. (Start address + Length) is not greater than 0xFFFFFFFF	4. Invalid omit
	5. Verify that omit identifier values are contained in two sets of braces	5. Invalid omit
	6. Repeat this test for each start address/length pair within the omit header field: If a memory address in the omit range overlaps with any byte of a memory area defined by a single address/length pair within erase header field, verify that start address/length pair in the omit header field exactly matches with start address/length pair in the erase header field.	6. Invalid omit
	7. Repeat this test for each start address/length pair within the omit header field: If a memory address in the omit range overlaps with any byte of a memory area defined by a single address/length pair within Data section, verify that start address/length pair in the omit header field exactly matches with start address/length pair in the Data section.	7. Invalid omit

	8. If omit values are preceded by '0x', verify that the alphanumeric characters are from the string "0123456789ABCDEFG"	8. Invalid omit
	9. If omit values are preceded by '0b', verify that the digits followed are either '0' or '1'	9. Invalid omit
	10. If omit values are not preceded by '0x' or '0b', verify that the characters followed are from 0 - 9	10. Invalid omit
180	Verify the following for call identifier (if present)	
	1. If sw_part_type is equal to "SBL" or "GBL", call identifier must be present	1. No call present
	2. If sw_part_type identifier is equal to "TEST", call identifier is optional	2. None
	3. If sw_part_type is not equal to SBL, GBL, or TEST, call identifier must not be present	3. call not allowed
	4. call identifier, if present, contains a start address four bytes long, ≥ 0, and ≤ 0xFFFFFFFF	4. Invalid call
	5. call identifier value is not contained in braces {}	5. Invalid call
	6. If call values are preceded by '0x', verify that the alphanumeric characters are from the string "0123456789ABCDEFG"	6. Invalid call
	7. If call values are preceded by '0b', verify that the digits followed are either '0' or '1'	7. Invalid call
	8. If call values are not preceded by '0x' or '0b', verify that the characters followed are from 0 - 9	8. Invalid call
190	Verify the following for file_checksum identifier:	
	1. file_checksum identifier contains a four byte file checksum	1. Invalid file_checksum
	2. file_checksum ≥ 0 and ≤ 0xFFFFFFFF	2. Invalid file_checksum
	3. Header file checksum matches actual data file checksum	3. Invalid file_checksum
	4. Initial value of file_checksum = 0xFFFFFFFF	4. Invalid file_checksum
	5. file_checksum identifier value is not contained in braces {}	5. Invalid file_checksum
	6. If file_checksum values are preceded by '0x', verify that the alphanumeric characters are from the string "0123456789ABCDEFG"	6. Invalid file_checksum
	7. If file_checksum values are preceded by '0b', verify that the digits followed are either '0' or '1'	7. Invalid file_checksum
	8. If file_checksum values are not preceded by '0x' or '0b', verify that the characters followed are from 0 - 9	8. Invalid file_checksum
200	Verify the following for Data Section:	
	1. Start address (first four bytes in each block) is ≥ 0 and ≤ 0xFFFFFFFF	1. Invalid address
	2. Length (four bytes following start address) is ≥ 0 and ≤ 0xFFFFFFFF	2. Zero_length
	3. (Start address + Length) is not greater than 0xFFFFFFFF	3. Address overflow, insufficient data
210	Verify the following for comments	
	1. Verify that for each begin comments character set (/*), there is an associated end comments character set (*/)	1 Invalid Comment
	2. Verify that begin and end comments character sets (/* and */) are not nested within another set	2 Invalid Comment

Notes:

1. Unless otherwise noted within this test specification, each issue shall result in an ERROR
2. This issue shall result in a WARNING