SOFTWARE REQUIREMENT DOCUMENT
FDEF

A/C APPLICABILITY	
ATA CHAPTER	42

REFERENCE	X42D06009883
ISSUE	17
DATE	10/09/2015

CUSTOMER	
CONFIDENTIALITY	

SUMMARY:

This document describes the software specifications adopted by the EYY Simulation Department for the development of the software part of the FDEF project.

FDEF is a software package generating the C code to format and deformat AFDX payloads, ARINC A429 labels, CAN messages and MIL1553 messages into discrete signals, integers or floats from Aircraft ICD and Model ICD type documents.

The FDEF project includes four parts corresponding to four separate tools:

- ICD2XML for the generation of xml files from AC/ICDs and MD/ICDs,
- GAC generating C code files from xml files.
- ADCN-Simu generating C code from ICD files to simulate switch failure
- ADCN-FDEF using the previous tools in hidden way to generate C code and a MICD file for a complete model

This document describes the specifications of these two parts.

KEYWORDS	FDEF, AFDX, A429, CAN, MIL1553, ADCN, ICD2XML, GAC, FDEFCollector, IFDEFIX, ADCN-FDEF, ADCN-Simu
RELATED DOCUMENTS	

	NAME	***SIGLUM - FUNCTION***	***DATE***	***SIGNATURE***
WRITTEN BY	M.BONNEVIALLE (SII)	Subcontractor for AIRBUS EYYSEW	10/09/2015	
APPROVAL	N.DELRIEU (AKKA)	Subcontractor for AIRBUS EYYQ		

	J-M.CLOUT	Simulation Framework Product Engineer		
AUTHORIZATION	F.GOUËZEC	Work Package Leader		

A-F DIDEROT ARCHIVING CARTOUCHE

REFERENCE :	X42D06009883	***ISSUE :***	17	***DATE :***	10/09/2015
TITLE :	FDEF SOFTWARE REQUIREMENT DOCUMENT				
SIGLUM :	EYYSER				
SECTION :					
WRITTEN BY :	M.BONNEVIALLE (SII)			***OWNER :***	J-M.CLOUT
PROGRAM :					
ATA CHAPTER :	42	***ATA SUBCHAPTER :***		***ATA SUBJECT :***	
PROJECT :	FDEF	***PROJECT REF :***		***REV :***	
CUSTOMER :					
KEYWORDS :	FDEF, AFDX, A429, CAN, MIL1553, ADCN, ICD2XML, GAC, FDEFCollector, IFDEFIX, ADCN-FDEF, ADCN-Simu				
RELATED DOCUMENTS :					
TYPE :	SOFTWARE REQUIREMENT DOCUMENT				
CONFIDENTIALITY :					
LANGUAGE :	English				

CANCEL PREVIOUS ISSUE :	Y	***PAGES TO BE ARCHIVED :***	100

<EXTERNAL DOCUMENT>					
ISSUING :					
REFERENCE :		***ISSUE :***		***DATE :***	

DISTRIBUTION LIST

DEPARTMENT/ COMPANY	NAME	P.O. BOX	COVER PAGE ONLY	NOTE WITHOUT ATTACH-MENT	NOTE WITH ATTACH-MENT
	On-line distribution				

RECORD OF REVISIONS

ISSUE	DATE	EFFECT ON PAGE	EFFECT ON PARA	REASONS FOR REVISION
01	08/03/2004	All		Creation of document
02	13/08/2004	5		Technical note describing the ATA42 monitor changes placed in applicable documents (see FDEF DMs 13 and14).
03	04/07/2006	All		ICD2XML part and FDEF CAN format and modification requests: FDEF DMs 30,31,32,37,38,41,42 and 44 taken into account.
04	16/10/2008	All		MIL1553 message format taken into account for GAC and ICD2XML tools.
05	02/08/2011			A429 opaques taken into account.
06	31/01/2012			FDEF DMs 128, 129, 165, 166, 167 and 172 taken into account.
07	20/06/2012			FDEF DMs 162 (/REFRESH_RATE_COEFF option) and 179 taken into account.
08	03/07/2012			Add of ADCN-Simu tool in FDEF
09	13/07/2012	82		Correction after review
10	17/07/2012	80, 81, 82	5.4.5 5.4.6	--target_p and --not_standalone options for ADCN-Simu Sorting rules for ADCN-simu
11	30/07/2012	16,17, 83-91	4.1 5.5 6.1 6.2.4	Add of ADCN-FDEF tool in FDEF
12	12/09/2012	16,47,81-90	4.1 5.2.7 5.4.5 5.4.6 5.5	Completion of ADCN-FDEF tool And add of the --fdef_names and --define_switch options for ADCN-Simu
13	06/12/2012	89, 91	5.5.6 5.5.8	Completion of ADCN-FDEF tool
14	07/04/2013	90,92	5.5.7	Addition of the –micd_m2633_2 of ADCN_FDEF
15	22/01/2014	63,75-76,82 43 76 91,93	5.3.4, 5.3.9, 5.4.5 5.2.8 5.3.9 5.5.7	Minor corrections through the document and news requirements regarding the code generated by ADCN-Simu /A429_LABEL_NAME_RULE2 by default for ICD2XML new options /IGNORE_FORMAT_INIT and /IGNORE_DEFORMAT_INIT for GAC new option --filt_network_B for ADCN-FDEF
16	30/01/2015	22-24 38-39 54 56-61 64-65 70 92-94	5.2.3 5.2.7 5.3.2 5.3.3 5.3.4 5.3.5 5.5.7	Minor corrections through the document Addition of the --xml_only option for ADCN-FDEF Modification of SSM treatment for A429 labels in ICD2XML and GAC Completion of char array initialisation specification in GAC

ISSUE	DATE	EFFECT ON		REASONS FOR REVISION
		PAGE	PARA	
17	10/09/2015	24 35-46 72 87-88 96	5.2.3 5.2.7 5.3.6 5.5.1 5.5.7	Minor corrections through the document Addition of the --keep_xml option for ADCN-FDEF Correction of SSM treatment for A429 HYB labels in ICD2XML and GAC

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1 Introduction

FDEF is a range of tools the aim of which is to generate code for the simulation models. This code is specifically dedicated to the formatting and deformatting of complex data (ARINC 429, AFDX, CAN, MIL1553) into simple data (integer, Boolean, float, etc.).

1.1 Purpose of document

The purpose of this document is to describe the software specifications of the ICD2XML, GAC, ADCN-Simu and ADCN-FDEF subsystems.

1.2 Scope

This document concerns the development of the FDEF software intended for the simulation products.

1.3 Applicable documents

Code	Reference	Title
DA1	L00D03016027	Format des fichiers XML en entrée de FDEF
DA2	AM2713	Normes de codages en C pour les modèles de simulation
DA3	L42SP0401855	Evolutions FDEF Moniteur ATA42
DA4	AM2771	Ingénierie des exigences

Table 1: Applicable documents

1.4 Reference documents

Code	Reference	Title
DR1	Airbus L42D1 515.1722/01 Issue 02, April 2002	AFDX Message Formats Detailed Functional Specification Airbus L42D1 515.1722/01 Issue 02, April 2002
DR2	1995	ARINC 429, Mark 33 Digital Information Transfer System (DITS)
DR3	357_L03SP0200051	A380 Spécification détaillée de la normalisation CTC
DR4	Airbus 515.0404/2002 Draft 3.1, 2002	Conversion & Routing Function Detailed Functional Specification
DR5	509.0381 3.3, 2002	A380 Simulation Models Interface Principle
DR6	L00ME0306858	Rules to populate the Data Base

Code	Reference	Title
DR7	M42SP0601538	Note technique : Traitement du format CAN par l'outil FDEF

Table 2: Reference documents

1.5 Glossary - Terminology

Term/Siglum	Meaning
AC/ICD	AirCraft ICD
ADCN	Advanced Data Communication Network
AFDX	Avionic Full DupleX Ethernet
AMO	Airbus MOdel
API	Application Programming Interface
ARINC	Aeronautical Radio INCorporation
ASPIC	Simulation workshop for integration and design
BDI	Interface database
CAN	Controller Area Network
DS	Data Set
DSS	Distributed Simulation Software
FDS	Functional Data Set
FDW	Functional Data Word
FS	Functional Status
GAC	Automatic Code Generator
ICD	Interface Computing Data
MD/ICD	MoDel ICD
MIL	MILitary
SSM	Sign/Status Matrix
TBD	To Be Defined

Table 3: Glossary

1.6 Requirement coding rules

This paragraph contains the rules adopted to identify the requirements.

1.6.1 Coding example

The requirements can be identified as follows:

E_Document_ [Type]_Index

The optional fields are between [ ].

Description of fields

Document: identifies the document (STL, DAS, DCP, PTV, etc.)

Type: short text to specify the function of the specified subsystem or the type of requirement

Index: chronological serial number (to be incremented in 10s to keep the possibility of inserting new requirements)

1.6.2 Traceability to inherited requirements

Not applicable.

2 Untraced upstream requirements

Not applicable.

3 Context and environment

The FDEF tool is an automatic code generator which analyses the information from the signal databases and generates formatting/deformatting code to be integrated into a simulation model.

The generated code allows so-called "preformatted" variables of integer, float or discrete type to be formatted as complex variables of AFDX, ARINC A429, CAN or MIL1553 type. Other files produced allow the reverse operation, deformatting, to be done.

Formatting and deformatting files are generated by means of Excel-type files `.csv` called AC/ICD and MD/ICD. These files describe all the AFDX, A429, CAN or MIL1553 messages and their preformatted ICD variables.

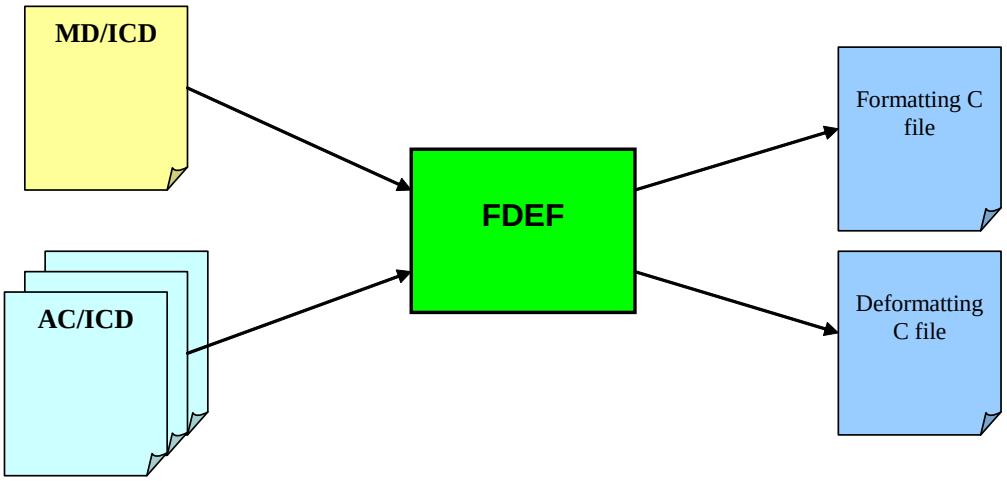

Figure 1: FDEF global operation

The C code files are generated by four tools:

- FDEF ICD2XML producing `.xml` files (called configuration tables) by analysing ICD files,
- FDEF GAC producing C code files by analysing xml configuration tables.
- FDEF ADCN-Simu producing C code files by analysing ICD files.
- FDEF ADCN-FDEF producing C code files and a MICD file by using the three other tools.

The main functions ensured by the code generated by FDEF are:

- Acquisition of simulated system information in "preformatted data" format
- Elaboration of AFDX payloads, ARINC 429 words, CAN messages and/or MIL1553 messages
- Provision of AFDX payloads, ARINC 429 words, CAN messages and/or MIL1553 messages
- Acquisition of AFDX payloads, ARINC 429 words, CAN messages and/or MIL1553 messages from other systems
- Extraction and conversion of preformatted data of AFDX payloads, ARINC 429 words, CAN messages and/or MIL1553 messages
- Provision of this extracted data

4 Functional description

4.1 Identification of functions and their interfaces

The ICD2XML function reads all the AC/ICDs and the model ICDs the formatting/deformatting code of which is to be generated. It produces eight XML files per model called: `AFDX_prod.xml`, `AFDX_xonso.xml`, `A429_prod.xml`, `A429_conso.xml`, `CAN_prod.xml`, `CAN_conso.xml`, `M1553_prod.xml` and `M1553_conso.xml`. These files are intended for the code generator: the GAC function.

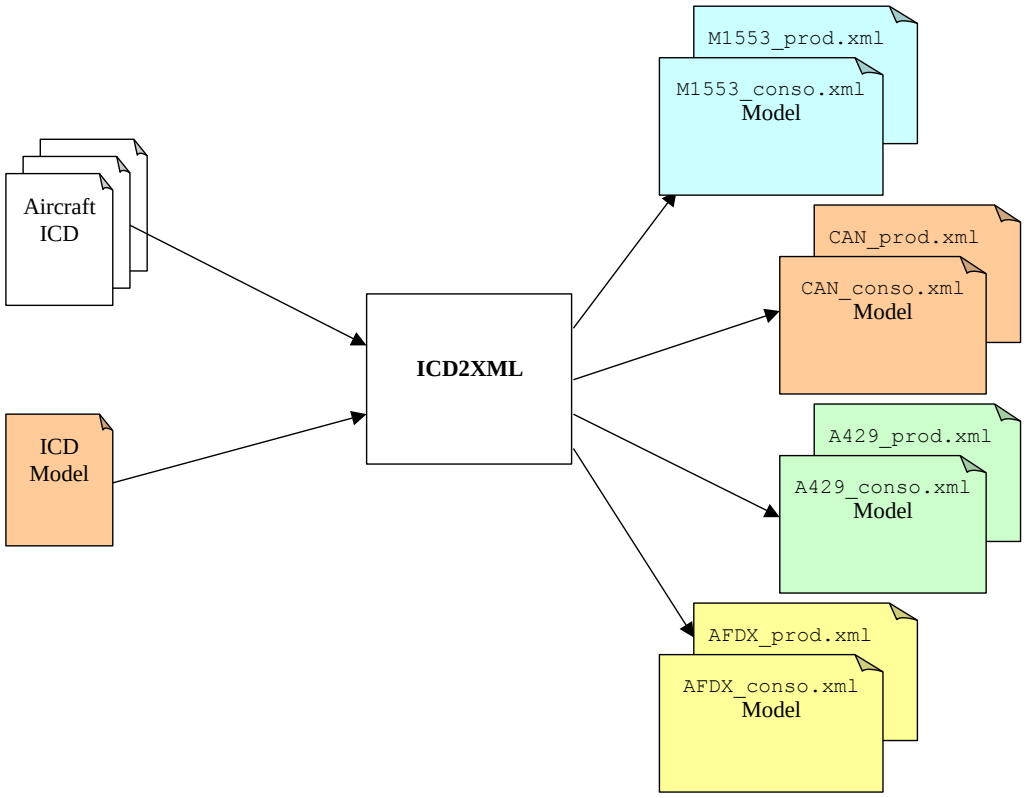

Figure 2: ICD2XML functionality

The GAC function takes at input the XML files describing the complex variables and their formatting according to preformatted variables. It generates the source code which can be integrated into an existing model or creates a model in its own right to ensure the formatting and deformatting of the data being simulated.

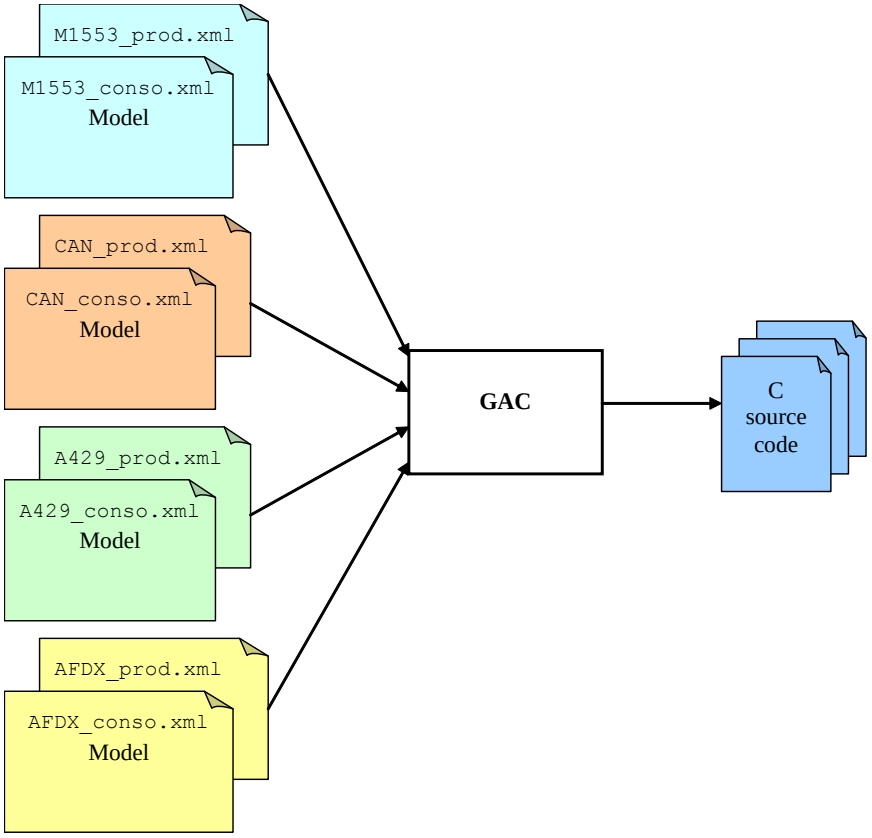

Figure 3: GAC functionality

The ADCN-Simu function reads all the AC/ICDs and the model ICD. It generates the source code which is used to simulate the ADCN switch failure.

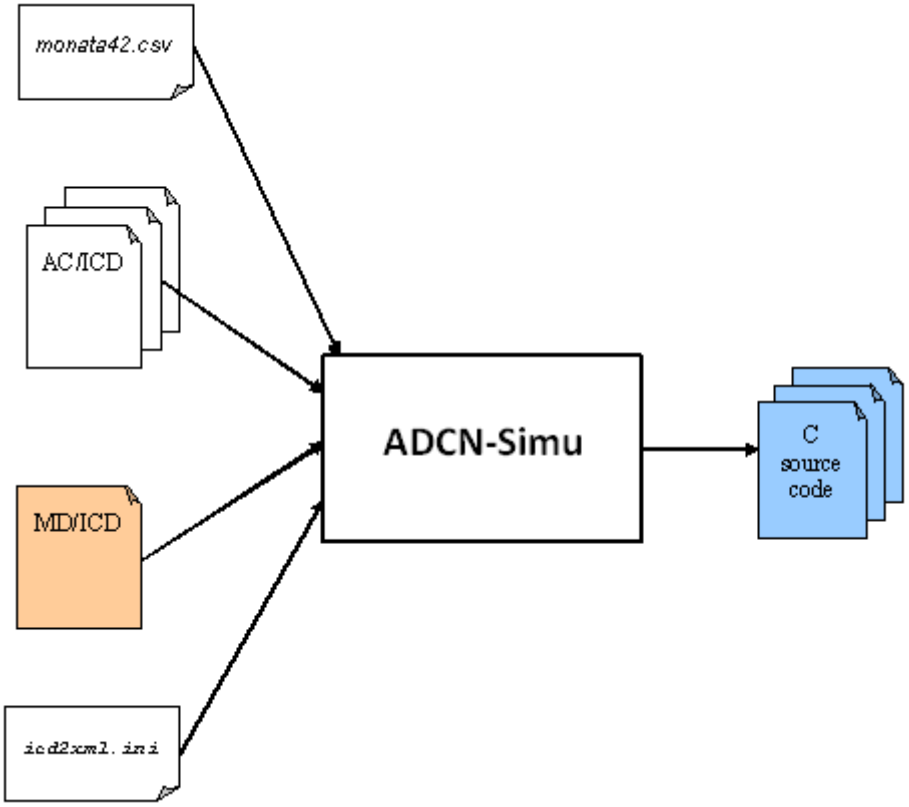

Figure 4 : ADCN-Simu functionality

The ADCN-FDEF function uses all the AC/ICDs, the model ICD and the ADCN file. It generates the source code which is used to simulate the ADCN switch failure and source code to format and deformat the model data by calling the other FDEF tools (ICD2XML, GAC and ADCN-Simu). It also generates a new model ICD listing the data used by the generated code and a main file that simulate the model by using the generated code.

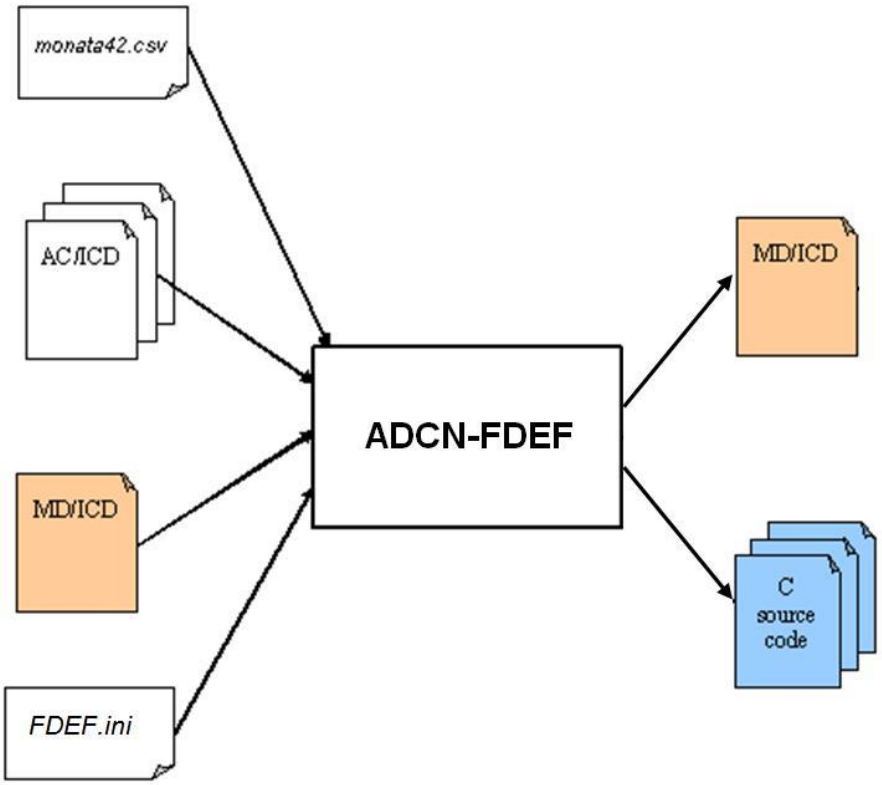

Figure 5 : ADCN-FDEF functionality

5 Functional specification

This paragraph details each of the functions described in the previous paragraph.

5.1 Transversal requirements

E_STL_FDEF_PERF_GAC_0010

On ASPIC 8.1.x and 8.2.x (running on DEC Alpha ES45), ASPIC 10.x (running on Linux PC) and on DSS (running on Intel Xeon 3.06 GHz bi-processor PC), the formatting time for 12000 elementary items of information into 250 AFDX payloads must not exceed 15 ms. Note that there are no upstream performance requirements on the formatting of ARINC words, CAN messages or MIL1553 messages.

#EndText
#Verify Test

E_STL_FDEF_PERF_GAC_0020

On ASPIC 8.1.x and 8.2.x (running on DEC Alpha ES45), ASPIC 10.x (running on Linux PC) and on DSS (running on an Intel Xeon 3.06 GHz bi-processor PC), the unformatting time for 250 AFDX payloads into 12000 elementary items of information must not exceed 15 ms. Note that there are no upstream performance requirements on the deformatting of ARINC words, CAN messages or MIL1553 messages.

#EndText

#Verify Test

5.2 ICD2XML function specification

This paragraph includes all the requirements relevant to the ICD2XML function.

5.2.1 ICD2XML function input files

E_STL_FDEF_ENT_ICD2XML_0010

Code generation is done by analysing the following files:

- AC/ICD containing the description of all the model variables (name, type, length and position in message, etc.); several AC/ICDs can be taken into account,
- MD/ICD containing the names of the signals and the key making the link with the associated message in the AC/ICD of AFDX, A429, CAN or MIL1553 type used for formatting,
- MD/ICD containing the names of the signals and the key making the link with the associated message in the AC/ICD of AFDX, A429, CAN or MIL1553 type used for deformatting.

#EndText
#Verify Test

5.2.2 Processing relevant to AFDX payloads

E_STL_FDEF_AFDX_ICD2XML_0010

ICD2XML processes the AFDX_INPUT_VL and AFDX_OUTPUT_VL lines of the AC/ICDs to get the identifier and the name of the AFDX payload.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0020

For each message, ICD2XML records the data contained in the following columns:

- `VL Identifier` (identifier included in the name of the AFDX payload)
- `VL name` (name associated with the AFDX payload)

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0030

ICD2XML processes the AFDX_INPUT_MESSAGE and AFDX_OUTPUT_MESSAGE lines of the AC/ICDs to get the list of AFDX payloads.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0040

For each message, ICD2XML records the data contained in the following columns:

- `Message name` (included in the name of the AFDX payload)
- `Associated VL` (link to find the "`VL Identifier`" of an AFDX payload by comparing it with the "`VL name`")
- `Message length` (size of the AFDX payload in bytes)
- `Port Refresh Rate` (refresh rate associated with the payload)
- `Message Transmission Rate` (transmission rate of the message)
- `Application name` (application associated with the payload)

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0050

The unique key which differentiates between two messages is given by:

```
Associated VL:Message name
```

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0060

ICD2XML processes the AFDX_INPUT_MESSAGE and AFDX_OUTPUT_MESSAGE lines of the AC/ICDs to get the list of parameters contained in the AFDX payloads.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0070

For each parameter of an AFDX payload, ICD2XML records the data contained in the following columns:

- `Parameter name` (name of parameter)
- `Parameter type` (type of parameter)
- `Parameter Keyword` (keyword to take the related parameter into account or not)
- `FDS name` (name of FDS)
- `FDS length` (size of FDS in bytes)
- `FDS address in the message` (address of FDS in bytes in the AFDX payload)
- `Functional Status name` (name of FS)
- `FS address in the message` (address of FS in bytes in AFDX payload)

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0080

The unique key which differentiates between two parameters within a given message is given in the "`Parameter name`" column.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0090

The types of parameters allowed for an AFDX payload are:

- Boolean
- Float
- Opaque
- Integer
- String

All parameters of other types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0100

ICD2XML processes the AFDX_INPUT_MESSAGE and AFDX_OUTPUT_MESSAGE lines of the AC/ICDs to get the list of signals contained in the AFDX parameters.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0110

For each signal, ICD2XML records the data contained in the following columns:

- `Signal name` (name of signal)
- `Signal Nb of bit` (number of signal bits)
- `Signal address` (address of the signal in bytes in the AFDX payload)
- `Signal position` (position of the signal in bits from "Signal address")
- `Float Coding type` (type of coding of float signals)
- `String length` (size of associated character string in bytes)
- Integer signed (indicates whether a BNR integer is signed)
- Float signed (indicates whether a BNR float is signed)

-

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0120

The unique key which differentiates between two signals is given by:

```
Associated VL:Signal name
```

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0130

The types of signals allowed are:

- Boolean
- Float
- Opaque
- Integer
- String

All signals of other types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ICD2XML_0140

If the column Port Refresh Rate is empty, its value is recomputed with this formula:

- Port Refresh Rate = round(Message Transmission Rate)

In which round returns the nearest tenth of the parameter with a minimum of 10.

#EndText
#Verify Test

5.2.3 Processing relevant to ARINC A429 labels

E_STL_FDEF_A429_ICD2XML_0010

ICD2XML processes the A429_INPUT_LABEL and A429_OUTPUT_LABEL lines of the AC/ICDs to get the list of ARINC A429 labels.

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0020

For each message, ICD2XML records the data contained in the following columns:

- `Application name` (name of the application associated with the A429 label)
- `Associated Bus` (name of the bus associated with the A429 label)
- `Label name` (included in the name of the A429 label)
- `Label Number` (label number included in the name of the A429 label)
- `Port Refresh Rate` (refresh rate associated with the A429 label)
- `Cycle Frequency` (frequency associated with the A429 label)
- `Label type` (type of label)
- `SDI` (SDI value of the A429 label)
- `val00Sdi` (value 0 of the SDI of the A429 label)
- `val01Sdi` (value 1 of the SDI of the A429 label)
- `val10Sdi` (value 2 of the SDI of the A429 label)
- `val11Sdi` (value 3 of the SDI of the A429 label)
- `SSM value state 00` (state of the A429 label when the SSM is equal to 00)
- `SSM value state 01` (state of the A429 label when the SSM is equal to 01)
- `SSM value state 10` (state of the A429 label when the SSM is equal to 10)
- `SSM value state 11` (state of the A429 label when the SSM is equal to 11)

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0030

The types of labels allowed are:

- BNR
- DIS
- BCD
- Opaque
- HYB
- ISO5

All labels of other types are considered as undefined.

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0035

On an "SDI" of an ARINC A429 label equal to "XX", the SDI parameter must be equal to:

- 00 if the "`Application name`" column corresponds to the "val00Sdi" column,
- 01 if the "`Application name`" column corresponds to the "val01Sdi" column,
- 10 if the "`Application name`" column corresponds to the "val10Sdi" column
- 11 if the "`Application name`" column corresponds to the "val11Sdi" column.

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0040

Type of parameter of the SSM of an A429 label is determined according to the "`Label type`" and "`SSM value state XX`" columns.

#EndText
#Verifiy Test

E_STL_FDEF_A429_ICD2XML_0041

If the "`Label type`" is equal do BNR, BCD or DIS, the type

of parameter of the SSM will be:
- for a BNR label : "`status_ssm_bnr`"
- for a BNR label : "`status_ssm_bcd`"
- for a BNR label : "`status_ssm_dis`"

#EndText
#Verifiy Test

E_STL_FDEF_A429_ICD2XML_0042

If the "`Label type`" is equal do HYB and the label contains either:
- one or several BNRs and one or several Booleans/Enumerates
- one or several BCDs and one or several Booleans/Enumerates

Then, the type of parameter of the SSM will be:
- if "`SSM value state 00`" is equal to FW : "`status_ssm_bnr`"
- else : "`status_ssm_bcd`"

In any other case, the type will be: "`status_no_ssm`"

#EndText
#Verifiy Test

E_STL_FDEF_A429_ICD2XML_0043

If the "`Label type`" is equal do Opaque or ISO5, the type of parameter of the SSM will be:

- if "SSM value state 00" is equal to NO, "SSM value state 01" is equal to NCD, "SSM value state 10" is equal to FT and "SSM value state 11" is equal to FW : "status_ssm_dis"
- if "SSM value state 00" is equal to FW, "SSM value state 01" is equal to NCD, "SSM value state 10" is equal to FT and "SSM value state 11" is equal to NO : "status_ssm_bnr"

In any other case, the type will be: "status_no_ssm"

#EndText
#Verifiy Test

E_STL_FDEF_A429_ICD2XML_0050

The unique key which differentiates between two messages is given by:

```
Associated Bus:Label name
```

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0060

ICD2XML processes the A429_INPUT_LABEL and A429_OUTPUT_LABEL lines of the AC/ICDs to get the list of parameters contained in the A429 label.

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0070

For each parameter, ICD2XML records the data contained in the following columns:

- Parameter name (name of parameter)
- Parameter type (type of parameter)

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0080

The unique key which differentiates between two parameters within a given message is given in the "Parameter name" column.

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0090

The types of parameters allowed are:

- Boolean
- Float
- Opaque

- Integer
- String

All parameters of other types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0100

ICD2XML processes the A429_INPUT_LABEL and A429_OUTPUT_LABEL lines of the AC/ICDs to get the list of signals contained in the A429 parameters.

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0110

For each signal of an ARINC A429 label, ICD2XML records the data contained in the following columns:

- `Signal name` (name of signal)
- `Signal Nb of bit` (number of signal bits)
- `Signal LSB` (position of LSB)
- `Signal MSB` (position of MSB)
- `Signal start bit` (start bit)
- `Float Coding type` (type of coding for float signals)
- `Float resolution` (resolution of float signals)
- `Integer Coding type` (type of coding of integer signals)
- `Integer resolution` (resolution of integer signals)
- Integer signed (indicates whether a BNR integer is signed)
- Float signed (indicates whether a BNR float is signed)

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0120

The unique key which differentiates between two signals is given by:

```
Associated Bus:Signal name
```

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0130

The types of signals allowed are:

- Boolean
- Float
- Opaque
- Integer
- String

All signals of other types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_A429_ICD2XML_0140

If the column Port Refresh Rate is empty, its value is recomputed with this formula:

- If (Cycle Frequency=0) Port Refresh Rate=100
- Else Port Refresh Rate = round(E(1000/Cycle Frequency))

In which round returns the nearest tenth of the parameter with a minimum of 10 and E returns the integer part of a number

#EndText
#Verify Test

5.2.4 Processing relevant to CAN messages

E_STL_FDEF_CAN_ICD2XML_0010

ICD2XML processes the CAN_INPUT_MESSAGE and CAN_OUTPUT_MESSAGE lines of the AC/ICDs to get the list of CAN messages.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0020

For each message, ICD2XML records the data contained in the following columns :

- `CAN Message name` (included in the CAN message name)
- `Port Refresh Rate` (refresh rate associated with the CAN message)
- `Msg Update Rate` (update rate of the message)
- `Associated Bus` (name of bus associated with the CAN message)

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0030

The unique key which differentiates between two messages is given by:

```
Associated Bus:CAN Message name
```

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0040

A message with a size different from 4 or 8 is ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0050

ICD2XML processes the CAN_INPUT_MESSAGE and CAN_OUTPUT_MESSAGE lines of the AC/ICDs to get the list of parameters contained in the CAN messages.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0060

For each parameter, ICD2XML records the data contained in the following columns

- `Parameter name` (name of parameter)
- `Parameter type` (type of parameter)

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0070

The unique key to differentiate between two parameters within a given message is given by the "Parameter name" column.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0080

The types of parameters allowed for the CAN are:

- Boolean
- Float
- Opaque
- Integer
- String

All parameters of other types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0090

ICD2XML processes the CAN_INPUT_MESSAGE and CAN_OUTPUT_MESSAGE lines of the AC/ICDs to get the list of signals contained in the CAN parameters.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0100

For each signal, ICD2XML records the data contained in the following columns:

- `Signal name` (name of signal)
- `Signal Nb of bit` (number of signal bits)
- `Signal LSB` (position of LSB)
- `Signal MSB` (position of MSB)
- `Signal start bit` (start bit)
- `Float Coding type` (type of coding for float signals)
- `Float resolution` (resolution of float signals)
- `Integer resolution` (type of coding for integer signals)
- `Integer Coding type` (resolution of integer signals)
- Integer signed (indicates whether a BNR integer is signed)
- Float signed (indicates whether a BNR float is signed)

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0110

The unique key which differentiates between two signals is given by:

```
Associated Bus:Signal name
```

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0120

The types of signals allowed are:

- Boolean

- Float
- Opaque
- Integer
- String

All signals of other types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0130

The values allowed for "`Float Coding type`" are:

- Simple Precision
- BCD
- BNR

All float signals of other coding types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0140

The values allowed for "`Integer Coding type`" are:

- BN2
- BCD
- BNR
- Opaque

All integer signals of other coding types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_CAN_ICD2XML_0150

If the column Port Refresh Rate is empty, its value is recomputed with this formula:

- Port Refresh Rate = round(Msg Update Rate)

In which round returns the nearest tenth of the parameter with a minimum of 10.

#EndText
#Verify Test

5.2.5 Processing relevant to MIL1553 messages

E_STL_FDEF_M1553_ICD2XML_0010

ICD2XML processes the MIL1553_MESSAGE lines of the AC/ICDs to get the list of MIL1553 messages.

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0020

For each message, ICD2XML records the data contained in the following columns:

- `Bus name` (included in the name of the MIL1553 messages)
- `Application name` (name of the application associated with the MIL1553 messages)
- `Message name` (name of the message associated with one or more MIL1553 messages)
- `Nb word` (number of Functional Data Words in a MIL1553 message)
- `Functional Data name` (name of the Functional Data Word in the MIL1553 message)
- `Functional Data address` (address of the Functional Data Word in the MIL1553 message in number of 16-bit words)
- `Functional Data length` (size of the Functional Data Word in the MIL1553 message in number of 16-bit words)

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0030

The unique key which differentiates between two MIL1553 messages is given by:

```
Application name:Message name
```

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0040

ICD2XML processes the MIL1553_INPUT_DATA and MIL1553_OUTPUT_DATA lines of the AC/ICDs to get the list of parameters contained in the MIL1553 messages.

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0050

The unique key which for a given parameter allows the information concerning the Bus and the Functional Data Word of the MESSAGE zone to be found is given by:

```
Application name:Message name:Functional Data name
```

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0060

For each parameter of a MIL1553 message, ICD2XML records the data contained in the following columns:

- `Application name` (name of the application intended for the joint key with the MESSAGE part)
- `Message name` (name of the message intended for the joint key with the MESSAGE part)
- `Functional Data name` (name of the Functional Data Word intended for the joint key with the MESSAGE part)
- `Parameter name` (name of parameter)
- `Parameter type` (type of parameter)
- `Parameter Keyword` (keyword to take the related parameter into account or not)

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0070

The types of parameters allowed for a MIL1553 message are:

- Boolean
- Float
- Opaque
- Integer
- String

All parameters of other types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0080

ICD2XML processes the MIL1553_INPUT_DATA and MIL1553_OUTPUT_DATA lines of the AC/ICDs to get the list of signals contained in the MIL1553 parameters.

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0090

For each signal, ICD2XML records the data contained in the following columns:

- `Signal name` (name of signal)
- `Signal Nb of bit` (number of signal bits)
- `Signal MSB` (position of MSB)
- `Float Coding type` (type of coding for float signals)
- `Float resolution` (resolution of float signals)
- `Integer Coding type` (type of coding for integer signals)
- `Integer resolution` (resolution of integer signals)

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0100

The unique key which allows the status of a MIL1553 message to be found is given by:

```
Bus name:Message name:Functional Data name
```

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0110

The unique key which differentiates between two signals is given by:

```
Bus name:Signal name
```

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0120

The types of signals allowed are:

- Boolean
- Float
- Opaque
- Integer
- String

All signals of other types are ignored and ICD2XML returns an error.

#EndText
#Verify Test

E_STL_FDEF_M1553_ICD2XML_0130

ICD2XML creates the name of the MIL1553 message from the "`Bus name`" and "`Message name`" fields of the AC/ICD according to the following principle:

ML_<Bus name>_<Message name>_I/O

#EndText
#Verify Test

5.2.6 Processing of MD/ICDs

E_STL_FDEF_MD_ICD2XML_0010

ICD2XML processes the lines of the MD/ICDs to get the list of formatted variables of the model.

The first MD/ICD passed by parameters is the one related to the AC/ICD INPUTS.

The second MD/ICD passed by parameters is the one related to the AC/ICD OUTPUTS.

#EndText
#Verify Test

E_STL_FDEF_MD_ICD2XML_0011

ICD2XML must ignore the queuing ports from the MD/ICDs identified by value "Q" in the comMode column. Only the sampling ports, associated with comMode value "S" must be processed by FDEF.

#EndText
#Verify Test

E_STL_FDEF_MD_ICD2XML_0020

For each variable, ICD2XML records the data contained in the following columns of the MD/ICD:

- `Name` (name of signal)
- `Type` (type of signal)
- `Unit` (signal units)
- `Description` (signal description)
- `Convention` (function applied to the signal before formatting or after deformatting)
- `Dim1` (size of the signal in bytes)
- `Refresh Rate` (signal refresh rate)
- `Com Format` (signal communication format)

- `Consumed If` or `Produced If` depending on whether the MICD is a model input or output MICD
- `Aircraft Signal Name` (joint key to relate the signal to the AC/ICD)
- `Status` (refresh signal or not)
- `Value` (signal initialisation value)
- `Interface Level`
- `Simulation Level`

#EndText
#Verify Test

E_STL_FDEF_MD_ICD2XML_0030

The unique key which differentiates between two variables within a given model is given by the "`Name`" column.

#EndText
#Verify Test

E_STL_FDEF_MD_ICD2XML_0040

Each variable of an MD/ICD is related to the AC/ICDs by means of the unique key entered into the "`Aircraft Signal Name`" column.

#EndText
#Verify Test

5.2.7 Files produced by ICD2XML

E_STL_FDEF_PROD_ICD2XML_0010

ICD2XML produces two files concerning the AFDX payloads:

- `AFDX_prod.xml` contains the AFDX payloads,
- `AFDX_conso.xml` contains the AFDX payloads.

#EndText
#Verify Analysis

E_STL_FDEF_PROD_ICD2XML_0020

#Verify Analysis

The format of the AFDX payload xml files is as follows:

#EndText
#Verify Analysis

```
<?xml version="1.0" standalone="yes" ?>
<configurationTable name="AFDX_TABLE" type="AFDX"
    generationDate="XX/XX/XXXX xxhxxm" generationTool="ICD2XML_V5.4">
  <configurationSources>
      <sourceFile type="modelICD" pathname="mdicd_input.csv" />
      <sourceFile type="aircraftICD" pathname="ac_icd.csv" />
  </configurationSources>
  <configurationEntity>
<message name="VL_1850_Message_Brake_Frame_120_1_O" length="64"
refreshRate="27">
      <FDS name="DS_1_Brake_Frame_120" address="8" length="8">
        <FS address="4">
          <parameter name="FS_1_Brake_Frame_120" type="status">
              <signal name="status_4301_1_1_prim_a"
                application="PRIM1A" comment="AFDX FS"/>
            </parameter>
          </FS>
        <parameter name="PRIMA_THSPOS_Brake_Frame_120_parameter"
            type="Float">
            <signal name="THSPOS_4301_1_prim_a" type="float" nbBit="32"
              address="8" init="N827402" application="PRIM1A"
              comment="THS position"/>
          </parameter>
        </FDS>
      <FDS name="DS_2_Brake_Frame_120" address="16" length="8">
        <FS address="5">
          <parameter name="FS_2_Brake_Frame_120" type="status">
              <signal name="status_4301_2_1_prim_a"
                application="PRIM1A" />
            </parameter>
          </FS>
          <parameter name="PRIMA_BFGPRIO_Brake_Frame_120_parameter"
            type="Boolean">
            <signal name="BFGPRIO_4301_1_prim_a" type="boolean"
              nbBit="1" address="16" position="1" init="N827411"
              application="PRIM1A" comment="PRIM Master for FGE"/>
          </parameter>
          <parameter name="PRIMA_B1APENGD_Brake_Frame_120_parameter"
            type="Boolean">
            <signal name="B1APENGD_4301_1_prim_a" type="boolean"
              nbBit="1" address="16" position="2" init="N827411"
              application="PRIM1A" comment="One AP engaged"/>
          </parameter>
          <parameter name="PRIMA_BNAV_Brake_Frame_120_parameter"
            type="Boolean">
            <signal name="BNAV_4301_1_prim_a" type="boolean" nbBit="1"
              address="16" position="3" init="N827411"
              application="PRIM1A" comment="NAV mode engaged"/>
          </parameter>
          <parameter name="PRIMA_BAP2ONLY_Brake_Frame_120_parameter"
            type="Boolean">
            <signal name="BAP2ONLY_4301_1_prim_a" type="boolean"
              nbBit="1" address="16" position="4" init="N827411"
              application="PRIM1A" comment="AP2 only is engaged"/>
          </parameter>
        </FDS>
      </message name>
  </configurationEntity>
```

Figure 6: Format of `AFDX_xxx.xml` files

Attribute	Information contained
message name	constructed from the "VL Identifier" and "Message name" columns of the AC ICD. VL_<VL Identifier>_<Message name>_I/O
message refreshRate	"Port Refresh Rate" column of the AC ICD If the previous parameter is empty : computation of the "Message Transmission Rate" column of the AC ICD
FDS name	"FDS name" column of the AC ICD.
FDS address	"FDS Address in the message" column of the AC ICD.
FDS length	"FDS length" column of the AC ICD.
FS address	"FS Address in the message" column of the AC ICD.
FS parameter name	"Functional Status name" column of the AC ICD.
FS parameter type	always equal to "status".
FS parameter signal name	"name" column of the MICD.
parameter name	"Parameter name" column of the AC ICD.
parameter type	"Parameter type" column of the AC ICD.
parameter signal name	"name" column of the MICD.
parameter signal type	"type" column of the MICD.
parameter signal nbBit	"Signal Nb of bit" column of the AC ICD.
parameter signal address	"Signal address" column of the AC ICD.
parameter signal position	"Signal position" column of the AC ICD.
parameter signal init	"Default Value" column of the MICD.
parameter signal application	"Application name" column of the AC ICD.

Attribute	Information contained
`Parameter signal comment`	"`Description`" column of the MICD.
`float floatCodingType`	"`Float Coding Type`" column of the AC ICD.
`string stringLength`	"`String length`" column of the AC ICD.
`Parameter signed`	Equal to 1 if: - "Integer signed" column = "yes" and "`Parameter type`" column = Integer - "Float signed" column = "yes" and "`Parameter type`" column = Float

Table 4: Correspondence of AFDX xml file flags

E_STL_FDEF_PROD_ICD2XML_0030

ICD2XML produces two files concerning the ARINC A429 labels:

- `A429_prod.xml` contains the ARINC A429 labels,
- `A429_conso.xml` contains the ARINC A429 labels.

#EndText
#Verify Analysis

E_STL_FDEF_PROD_ICD2XML_0040

The format of the xml files of the ARINC A429 labels is as follows:

#EndText
#Verify Analysis

```
<?xml version="1.0" standalone="yes" ?>
<configurationTable name="A429_TABLE" type="A429"
    generationDate="XX/XX/XXXX xxhxxm" generationTool="ICD2XML_V5.4">
  <configurationSources>
      <sourceFile type="modelICD" pathname="mdicd_input.csv" />
      <sourceFile type="aircraftICD" pathname="ac_icd.csv" />
  </configurationSources>
  <configurationEntity>
    <A429Label name="A4Label_1_O" type="BNR" refreshRate="60" labelNumber="XXX"
sdi="XX">
        <ssm type="status_ssm_bnr">
        <parameter name="Param_1_SSM" type="status">
            <signal name="signal_1_status" comment="NA"/>
         </parameter>
        </ssm>
       <parameter name="Param_float" type="Float">
        <signal name="signal_float" type="float" nbBit="15" lsb="14" msb="28"
            startBit="1" signed="1" comment="NA">
            <float floatResolution="0.00390" floatCodingType="BNR" />
          </signal>
      </parameter>
    </A429Label>
    <A429Label name="A4Label_2_O" type="Discrete" refreshRate="60"
labelNumber="270" sdi="XX">
      <ssm type="status_ssm_dis">
        <parameter name="Param_2_SSM" type="status">
            <signal name=" signal_2_status" comment="NA"/>
          </parameter>
        </ssm>
      <parameter name="Param_bool_1" type="logical">
          <signal name="signal_bool_1" type="boolean" nbBit="1" lsb="12"
            msb="12" startBit="1" init="0" comment="NA"/>
        </parameter>
      <parameter name="Param_bool_2" type="logical">
          <signal name="signal_bool_2" type="boolean" nbBit="1" lsb="25"
            msb="25" startBit="1" init="0" comment="NA"/>
        </parameter>
    </A429Label>
    <A429Label name="A4Label_3_O" type="BCD" refreshRate="60" labelNumber="XXX"
sdi="10">
        <ssm type="status_ssm_bcd">
        <parameter name="Param_3_SSM" type="status">
            <signal name="signal_3_status" comment="NA"/>
         </parameter>
        </ssm>
       <parameter name="Param_int" type="Integer">
        <signal name="signal_int" type="int" nbBit="15" lsb="14" msb="28"
            startBit="1" comment="NA">
            <integer integerResolution="0.00390" integerCodingType="BCD" />
          </signal>
      </parameter>
    </A429Label>
```

Figure 7: Format of `A429_xxx.xml` files

Attribute	Information contained
A429Label name	constructed from the "Associated Bus" and "Label Number" columns of the AC ICD. A4<Associated Bus>< Label Number>_<I/O>
A429Label refreshRate	"Port Refresh Rate" column of the AC ICD If the previous parameter is empty: computation of the "Cycle Frequency" column of the AC ICD
A429Label type	corresponds to the content of the "Label type" column of the AC ICD.
A429Label labelNumber	corresponds to the content of the "Label Number" column of the AC ICD.
A429Label sdi	corresponds to the content of the "SDI" column of the AC ICD.
ssm type	equal to "status_ssm_bcd", "status_ssm_bnr", "status_ssm_dis" or "status_no_ssm"
ssm parameter name	constructed from the "Label name" column of the AC ICD. <Label name>_SSM
ssm parameter type	equal to "status"
ssm parameter signal name	"name" column of the MICD.
parameter name	"Parameter name" column of the AC ICD.
parameter type	"Parameter type" column of the AC ICD.
parameter comment	"Description" column of the MICD.
parameter signal name	"name" column of the MICD.
parameter signal type	"type" column of the MICD.
parameter signal nbBit	"Signal Nb of bit" column of the AC ICD.
parameter signal lsb	"Signal LSB" column of the AC ICD.
parameter signal msb	"Signal MSB" column of the AC ICD.

Attribute	Information contained
parameter signal startBit	"Signal Start bit" column of the AC ICD.
parameter signal init	"Default Value" column of the MICD.
Parameter signal comment	"Description" column of the MICD.
float floatCodingType	"Float Coding type" column of the AC ICD.
float floatResolution	"Float Resolution" column of the AC ICD.
integer integerCodingType	"Integer Coding type" column of the AC ICD.
integer integerResolution	"Integer Resolution" column of the AC ICD.
Parameter signed	Equal to 1 if: - "Integer signed" column = "yes" and "Parameter type" column = Integer - "Float signed" column = "yes" and "Parameter type" column = Float

Table 5: Correspondence of A429 xml file flags

E_STL_FDEF_PROD_ICD2XML_0050

ICD2XML produces two files concerning the CAN messages:

- CAN_prod.xml contains the CAN messages,
- CAN_conso.xml contains the CAN messages.

#EndText
#Verify Analysis

E_STL_FDEF_PROD_ICD2XML_0060

The format of the xml files of the CAN messages is as follows:

#EndText
#Verify Analysis

```
<?xml version="1.0" standalone="yes" ?>
<configurationTable name="CAN_TABLE" type="CAN" generationDate="XX/XX/XXXX
    xxhxxm" generationTool="ICD2XML_V5.4">
  <configurationSources>
      <sourceFile type="modelICD" pathname="mdicd_input.csv" />
      <sourceFile type="aircraftICD" pathname="ac_icd.csv" />
  </configurationSources>
  <configurationEntity>
    <message name="CN_XXX_1_I" refreshRate="100">
      <status>
        <parameter name="Param_status_1" type="status">
            <signal name="signal_status_1" />
          </parameter>
      </status>
      <parameter name="Param_bool_1" type="Boolean">
          <signal name=" signal_bool_1" type="booleen" nbBit="1"
            lsb="45" msb="45" startBit="1" init="0" />
      </parameter>
      <parameter name=" Param_bool_2" type="Boolean">
          <signal name="signal_bool_2" type="booleen" nbBit="1"
            lsb="46" msb="46" startBit="1" init="0" />
      </parameter>
      <parameter name=" Param_bool_3" type="Boolean">
          <signal name="signal_bool_3" type="booleen" nbBit="1"
            lsb="51" msb="51" startBit="1" init="0" />
      </parameter>
      <parameter name=" Param_bool_4" type="Boolean">
          <signal name="signal_bool_4" type="booleen" nbBit="1"
            lsb="52" msb="52" startBit="1" init="0" />
      </parameter>
      <parameter name=" Param_bool_5" type="String">
          <signal name="signal_bool_5" type="string" nbBit="40"
            lsb="1" msb="40" startBit="1" init="0" />
      </parameter>
    </message>
    <message name="CN_XXX_2_I" refreshRate="100">
      <status>
        <parameter name="Param_status_2" type="status">
            <signal name="signal_status_2" />
          </parameter>
      </status>
      <parameter name=" Param_int" type="Integer">
          <signal name=" signal_int" type="int" nbBit="32" lsb="9" lsb="40"
comment="NA">
          <integer integerResolution="1" integerCodingType="BCD" />
          </signal>
        </parameter>
    </message>
  </configurationEntity>
```

Figure 8: Format of `CAN_xxx.xml` files

Attribute	Information contained
message name	constructed from the "Associated Bus" and "CAN Message name" columns of the AC ICD. CN_<Associated Bus>_<message>_<I/O>
message refreshRate	"Port Refresh Rate" column of the AC ICD If the previous parameter is empty: computation of the "Msg Update Rate" column of the AC ICD
status parameter name	constructed from the "Associated Bus" and "CAN Message name" columns of the AC ICD. CN_< Associated Bus >_<message>_Status
status parameter type	always equal to "status".
status parameter signal name	"name" column of the MICD.
parameter name	"Parameter name" column of the AC ICD.
parameter type	"Parameter type" column of the AC ICD.
parameter signal name	"name" column of the MICD.
parameter signal type	"type" column of the MICD.
parameter signal nbBit	"Signal Nb of bit" column of the AC ICD.
parameter signal lsb	"Signal LSB" column of the AC ICD.
parameter signal msb	"Signal MSB" column of the AC ICD.
parameter signal startBit	"Signal Start bit" column of the AC ICD.
parameter signal init	"Default Value" column of the MICD.
Parameter signal comment	"Description" column of the MICD.
float floatResolution	"Float Resolution" column of the AC ICD.
float floatCodingType	"Float Coding type" column of the AC ICD.
integer integerResolution	"Integer Resolution" column of the AC ICD.
integer integerCodingType	"Integer Coding type" column of the AC

Attribute	Information contained
	ICD.
Parameter signed	Equal to 1 if: - "Integer signed" column = "yes" and "Parameter type" column = Integer - "Float signed" column = "yes" and "Parameter type" column = Float

Table 6: Correspondence of CAN xml file flags

E_STL_FDEF_PROD_ICD2XML_0070

ICD2XML produces two files concerning the MIL1553 messages:

- M1553_prod.xml contains the formatting MIL1553 messages,
- M1553_conso.xml contains the deformatting MIL1553 messages.

#EndText
#Verify Analysis

E_STL_FDEF_PROD_ICD2XML_0080

The format of the xml files of the MIL1553 messages is as follows:

#EndText
#Verify Analysis

```
<?xml version="1.0" standalone="yes" ?>
<configurationTable name="M1553_TABLE" type="M1553" generationDate="XX/XX/XXXX
    xxhxxm" generationTool="ICD2XML_V6.0">
  <configurationSources>
      <sourceFile type="modelICD" pathname="mdicd_input.csv" />
      <sourceFile type="aircraftICD" pathname="ac_icd.csv" />
  </configurationSources>
  <configurationEntity>
    <message name="ML_XXX_Message1_O" nbWord="3">
      <FDW name="Word1_float" address="0" length="1">
        <status>
            <parameter name="Param_1_Status" type="status">
                  <signal name="signal_status_1" init="0" application="APP1" />
            </parameter>
        </status>
        <parameter name="Param_float" type="Float">
            <signal name="signal_float" type="float" nbBit="16" address="0"
              msb="16" init="1.0" application="APP1">
            <float floatResolution="0.00390" floatCodingType="BNR"/>
            </signal>
        </parameter>
     </FDW>
     <FDW name="Word2_discret" address="1" length="1">
       <status>
            <parameter name=" Param_2_Status" type="status">
                  <signal name="signal_status_2" init="0" application="APP1" />
            </parameter>
        </status>
        <parameter name="Param_bool_2" type="Boolean">
            <signal name="signal_bool_2" type="logical" nbBit="1" address="1"
              msb="3" init="" application="APP1">
            </signal>
        </parameter>
        <parameter name="Param_bool_5" type="Boolean">
            <signal name="signal_bool_5" type="logical" nbBit="1" address="1"
              msb="14" init="" application="APP1">
            </signal>
        </parameter>
        <parameter name="Param_bool_6" type="Boolean">
            <signal name="signal_bool_6" type="logical" nbBit="1" address="1"
              msb="16" init="" application="APP1">
            </signal>
        </parameter>
     </FDW>
     <FDW name="Word3_int" address="3" length="1">
       <status>
            <parameter name=" Param_3_Status" type="status">
                  <signal name="signal_status_3" comment="NA"/>
            </parameter>
       </status>
       <parameter name=" Param_int" type="Integer">
            <signal name="signal_int" " type="int" nbBit="12" address="2"
              msb="14" init="" application="APP1">
            <integer integerResolution="0.0460" integerCodingType="BNR"/>
            </signal>
       </parameter>
     </FDW>
    </message>
  </configurationEntity>
</configurationTable>
```

Figure 9: Format of `M1553_xxx.xml` files

Attribute	Information contained
message name	constructed from the "Bus name" and "Message name" columns of the AC ICD. ML_<Bus name>_<Message name>_I/O
message nbWord	"Nb word" column of the AC ICD.
FDW name	"Functional Data name" column of the AC ICD.
FDW address	"Functional Data address" column of the AC ICD.
FDW length	"Functional Data length" column of the AC ICD.
status parameter name	constructed from the "Functional Data name" column of the AC ICD. <Functional Data name>_Status
status parameter type	always equal to "status".
status parameter signal name	"name" column of the MICD.
parameter name	"Parameter name" column of the AC ICD.
parameter type	"Parameter type" column of the AC ICD.
signal name	"name" column of the MICD.
signal type	"type" column of the MICD.
signal nbBit	"Signal Nb of bit" column of the AC ICD.
signal msb	"Signal MSB" column of the AC ICD.
signal init	"Default Value" column of the MICD.
signal comment	"Description" column of the MICD.
signal application	"Application name" column of the AC ICD.
float floatResolution	"Float Resolution" column of the AC ICD.

Attribute	Information contained
`float floatCodingType`	"`Float Coding type`" column of the AC ICD.
`integer integerResolution`	"`Integer Resolution`" column of the AC ICD.
`integer integerCodingType`	"`Integer Coding type`" column of the AC ICD.

Table 7: Correspondence of MIL1553 xml file flags

E_STL_FDEF_PROD_ICD2XML_0090

ICD2XML produces the deltaICD.csv file listing all of the formatted refreshes and ports (initially present in the MICDs or created by FDEF) handled by FDEF. It obeys the same schema as an MICD csv except for the Direction column.

#EndText
#Verify Analysis

E_STL_FDEF_PROD_ICD2XML_0100

The deltaICD.csv file must reproduce, for each formatted refresh or port described, the following information:

Attribute	Information contained
Direction	"Input" if the corresponding port must generate a deformatting code, "Output" if it must generate a formatting code.
Name	Name of the port according to the FDEF naming rule (name of the port from the MICD for /MICD_NAMES option).
Type	Depends on the type of data used.
Unit	"Unit" column of the MICD if information available, else "wu".
Description	Name of the port from the MICD if the port already exists in the MICD. If the port was created by FDEF, or if the /MICD_NAMES option is used, the name is copied according to the FDEF naming rule.
Dim1	Size of formatted port.
Dim2	1
Com Format	Communication format associated with the port (AFDX, ARINC, CAN or M1553).

Attribute	Information contained
Com Mode	"S" (queuing not yet processed)
Refresh Rate	"Refresh Rate" column of the MICD (last line associated with this formatted port).
Aircraft Signal Name	Joint key of the port.
Interface Level	"Interface Level" column of the MICD (last line associated with this formatted port).
Status	"False" for a message, "True" for a refresh.
Consumed If / Produced If	"Consumed If" (fun_in) or "Produced If" (fun_out) column of the MICD (last line associated with this formatted port).
Simulation Level [1]	"Simulation Level [1]" column of the MICD (last line associated with this formatted port).

Table 8: List of attributes completed in deltaICD.csv file

#EndText
#Verify Analysis

5.2.8 ICD2XML command

E_STL_FDEF_CMD_ICD2XML_0010

The command line for the execution of ICD2XML includes the path of a temporary file including the paths of all the AC/ICDs used, the paths of the MD/ICDs and the generation options.

The format of the command line is as follows:

```
icd2xml <AC/ICD description file> <MD/ICD 1 file> <MD/IC 2 file>
<options>
```

#EndText
#Verify Test

E_STL_FDEF_CMD_ICD2XML_0020

ICD2XML accepts the options concerning the direction of the following MD/ICDs:

- `/INPUT_FORMAT`: the INPUT messages of the AC/ICDs are formatted and the OUTPUTS deformatted (by default).
- `/INPUT_DEFORMAT`: the OUTPUT messages of the AC/ICDs are formatted and the INPUTS deformatted.

#EndText

#Verify Test

E_STL_FDEF_CMD_ICD2XML_0030

ICD2XML accepts the options concerning the following joint keys:

- `/VL_SIGNAL`: key of "`Signal name`" = <`Associated VL`>:<`Signal name`> type for the AFDX payloads
- `/SIGNAL_ONLY`: key of "`Signal name`" = <`Signal name`> type
- `/SIGNAL_AMO`: single, combined or multiple key (by default)

#EndText
#Verify Test

E_STL_FDEF_CMD_ICD2XML_0040

ICD2XML accepts the options concerning the type of data to be formatted or deformatted (not exclusive):

- `/AFDX`: the AFDX payloads are taken into account
- `/A429`: the ARINC A429 labels are taken into account
- `/CAN`: the CAN messages are taken into account
- /M1553: the MIL1553 messages are taken into account

#EndText
#Verify Test

E_STL_FDEF_CMD_ICD2XML_0050

ICD2XML accepts the options concerning the names of the ARINC 429 labels:

- `/A429_LABEL_NAME_RULE0`: name of the ARINC A429 label according to standard: A4<Associated Bus><Label Number>
- `/A429_LABEL_NAME_RULE1`: name of the ARINC A429 label according to standard: <Label name>
- `/A429_LABEL_NAME_RULE2`: name of the ARINC A429 label according to standard: A4<Associated Bus><Label Number><SDI> (by default)

#EndText
#Verify Test

E_STL_FDEF_CMD_ICD2XML_0060

ICD2XML accepts the options concerning the lines of the AC/ICDs to be taken into account for the AFDX payloads:

- `/VL_PARAM_KEY` <keyword>: AC/ICD lines taken into account if the character string of the "Parameter Keyword" column is empty or identical to the "keyword" parameter.

#EndText

#Verify Test

E_STL_FDEF_CMD_ICD2XML_0070

ICD2XML accepts the options concerning the names of the formatted ports and refreshes copied into the deltaICD.csv file:

- `/MICD_NAMES`: inverts the contents of the Name and Description columns in the deltaICD.csv output file. The Name column then contains, for the formatted ports and refreshes already present in the MICD, the name from the MICD and not the one corresponding to the FDEF naming rule.

#EndText
#Verify Test

5.3 Specification of GAC function

5.3.1 Input files of GAC function

E_STL_FDEF_ENT_GAC_0010

The code is generated by analysing the following configuration files:

- `AFDX_prod.xml` for the description of the AFDX payloads to be produced
- `A429_prod.xml` for the description of the ARINC 429 labels to be produced
- `CAN_prod.xml` for the description of the CAN messages to be produced
- `M1553_prod.xml` for the description of the MIL1553 messages to be produced
- `AFDX_conso.xml` for the description of the AFDX payloads to be consumed
- `A429_conso.xml` for the description of the ARINC 429 labels to be consumed
- `CAN_conso.xml` for the description of the CAN messages to be consumed
- `M1553_conso.xml` for the description of the MIL1553 messages to be consumed

The format for these XML tables is given in DA1.

#EndText
#Verify Test

5.3.2 Processing relevant to AFDX payloads

In the remainder of this document, the LSB is numbered 0 and the MSB is represented by character "n" corresponding to the length of the AFDX payload. The following requirements

describe that which the generated code must ensure for the formatting or deformatting of the AFDX payloads.

E_STL_FDEF_AFDX_GAC_0010

The code generated by GAC must ensure the following conversions for the AFDX payloads:

- AFDX payload <–> float
- AFDX payload <–> integer
- AFDX payload <–> Boolean
- AFDX payload <–> character string
- AFDX payload <–> opaque

In the generated code, the Boolean type shall be an integer where value 0 shall correspond to FALSE and the other values to TRUE.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0020

The name of the formatted payload is given by the "name" attribute of the "message" flag of the `AFDX_prod.xml` and `AFDX_conso.xml` files.

The type of the variable is a char table the size of which is given by the "length" attribute of the "message" flag.

The name of a preformatted variable is given by the "name" attribute of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0030

The FSs of an AFDX payload represent the validity of an FDS. The FSs corresponding to status signals are defined as follows:

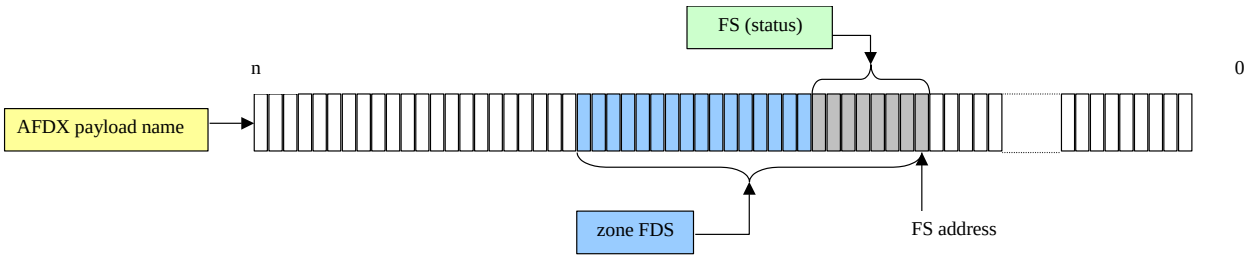

Figure 10: Location of a validity signal in an AFDX payload

The FS address value is given by the "address" field of the "FS" flag and represents the number of the byte from which the data is found. An FS is coded over one byte in the AFDX payload.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0040

The coding of an FS according to the status signal is done according to the table below:

#EndText
#Verify Test

FS	Designation	Simulated value	Meaning
0	Not Display (N.D.)	32	The data cannot be displayed.
0	Failure Warning (FW)	8	The data is invalid.
48	Not Computed Data (N.C.D.)	4	The emitter does not guarantee the data.
12	Functional Test (F.T.)	16	The data does not represent a valid value for the aircraft.
0	Not Emitted (N.E.)	2	The data is not emitted.
3	Normal Operation (N.O.)	0	The data is valid.

Table 9: FS correspondence

E_STL_FDEF_AFDX_GAC_0045

The FDEF GAC produces a Boolean type variable for the Refresh status of the AFDX message the name of which is constructed from the "name" field of the "message" flag of the xml: <message name>_R files.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0046

The value of the refresh Boolean on formatting depends on the conditions on the status signals of the AFDX frame:

#EndText
#Verify Test

Refresh value	Input parameter	Condition
1	N/A	No status signal value of the AFDX frame is set to 0 (Not Emitted).
0	/REFRESH_LOG0	At least one status signal of the AFDX frame is set to 2 (Not Emitted).

Refresh value	Input parameter	Condition
0	/REFRESH_LOG1	All the status signals of the AFDX frame are set to 2 (Not Emitted).

Table 10: Condition on AFDX formatting Refresh value

E_STL_FDEF_AFDX_GAC_0047

The values of the status signals on deformatting depend on the conditions on the refresh Boolean of the AFDX frame:

#EndText
#Verify Test

Refresh value	Condition
1	All the status signals are converted according to the FS values if the refresh Boolean of the AFDX frame is equal to 1.
0	All the statuses are set to value 2 (Not Emitted) if the refresh Boolean of the AFDX frame is equal to 0.

Table 11: Condition on AFDX deformatting status values

E_STL_FDEF_AFDX_GAC_0050

If the validity of the AFDX payload does not exist in the configuration tables, the AFDX data is valid by default.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0060

The signals of Boolean type coded in an AFDX payload are defined as follows:

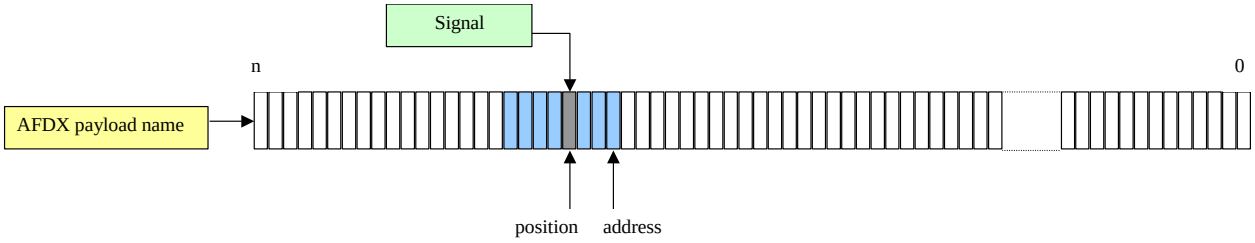

Figure 11: Location of a Boolean signal in an AFDX payload

The address value is given by the "address" field of the "signal" flag and represented by the number of the byte from which the data is found. The position value is given by the "position" field of the "signal" flag and represents the bit number in the data byte. It must be between 0 and 31.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0070

Signals of float type are coded in an AFDX payload as follows:

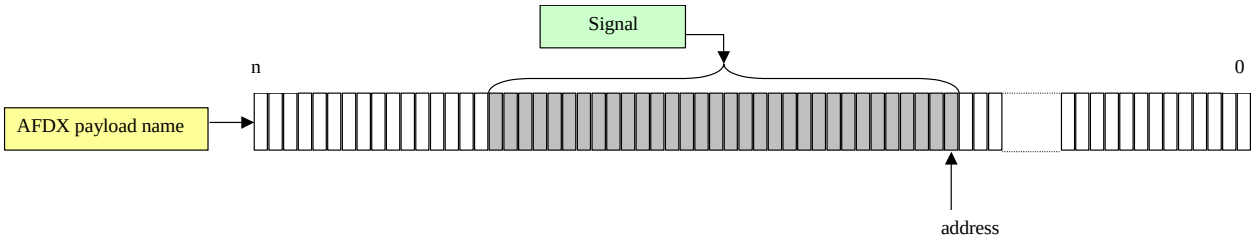

Figure 12: Location of a float signal in an AFDX payload

The address value is given by the "address" field of the "signal" flag and represents the number of the byte from which the data is found. The float signal is coded over 32 bits in an AFDX payload.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0080

The signals of integer type are coded in an AFDX payload as follows:

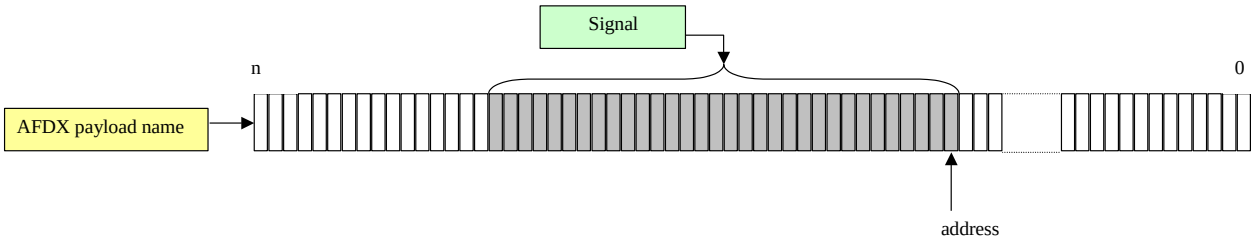

Figure 13: Location of an integer signal in an AFDX payload

The address value is given by the "address" field of the "signal" flag and represents the number of the byte from which the data is found. The integer signal is coded over 32 bits in an AFDX payload.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0090

The signals of "Opaque" and "String" type coded in an AFDX payload are defined as follows:

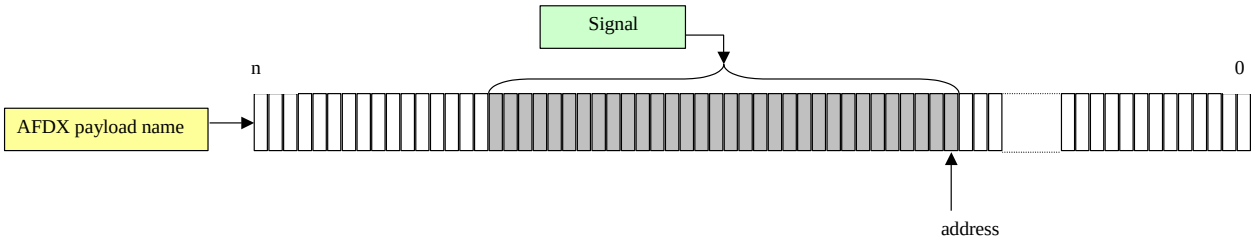

Figure 14: Location of an opaque or string signal in an AFDX payload

The address value is given by the "address" field of the "signal" flag and represents the number of the byte from which the data is found. The value of the signal is copied without any transformation or formatting over 32 bits.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0100

The AFDX preformatted variables are initialised if the "init" field of the "signal" flag is completed by a coherent value.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0105

For char array signals, the “init” field can be filled with several forms as illustrated in the following example.

The string ABBCCC can be written as:
- ABBCCC
- A/B/B/C/C/C
- ‘A’/’B’/’B’/’C’/’C’/’C’
- A/B*2/C*3
- ‘A’/’B’*2/’C’*3

The “/” character shall be escaped with a “\” if it is not a delimiter.

#EndText
#Verify Test

E_STL_FDEF_AFDX_GAC_0110

The FDEF GAC applies a conversion function given by the "convention" field of the "signal" flag during the formatting of a signal or the deformatting of an AFDX payload if it is not empty.

#EndText
#Verify Test

5.3.3 Processing relevant to ARINC A429 labels

In the remainder of this document, the LSB is numbered 0, the MSB is numbered 31. The following requirements describe that which the generated code must ensure for the formatting or deformatting of the ARINC A429 labels.

E_STL_FDEF_A429_GAC_0010

The code generated by GAC must ensure the following conversions for the ARINC A429 labels:

- ARINC 429 label <–> float
- ARINC 429 label <–> integer
- ARINC 429 label <–> Boolean
- ARINC 429 label <–> character string
- ARINC 429 label <–> opaque

In the generated code, the Boolean type shall be an integer the value 0 of which shall correspond to FALSE and the other values to TRUE.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0020

The name of the formatted ARINC A429 label is given by the "name" attribute of the "A429Label" flag of the `A429_prod.xml` and `A429_conso.xml` files.

The type of variable is a 32-bit "int" integer.

The name of a preformatted variable is given by the "name" attribute of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0030

The number of the label is given by the "labelNumber" field of the "A429Label" flag. This number is coded in inverted octal from bits 0 to 7 in all labels:

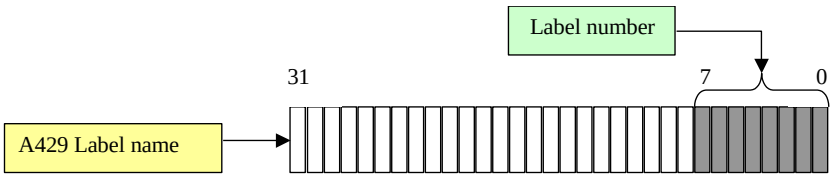

Figure 15: Location of label number

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0040

The type of ARINC A429 label defined by the "type" field of the "A429Label" flag of the `.xml` file determines the coding of the SSM of the ARINC A429 label and the type of coding of the signals in the label.

The SSM representing the validity of the ARINC A429 label is coded over two bits 30 and 31:

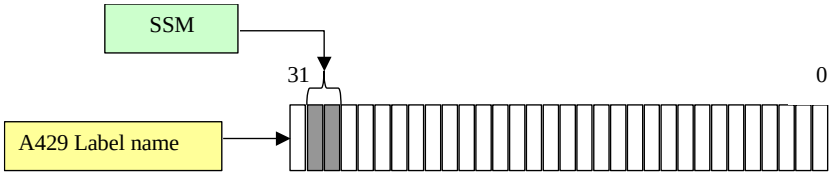

Figure 16: Location of validity signal in an A429 label

The name of the signal allowing the SSM to be coded is given by the "name" field of the "ssm" flag.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0050

The coding of the SSM according to the status signal for a BNR type label is done according to the table below (the "`type`" flag of the SSM parameter shall be equal to `status_ssm_bnr)`:

#EndText
#Verify Test

Bit 31	Bit 30	Designation	Simulated value	Meaning
0	0	Failure Warning (F.W.)	8	The data is invalid.
0	1	Not Computed Data (N.C.D.)	4	The emitter does not guarantee the data.
1	0	Functional Test (F.T.)	16	The data does not represent a valid value for the aircraft.
1	1	Normal Operation (N.O.)	0	The data is valid.

Table 12: Correspondence of SSM of a BNR label

E_STL_FDEF_A429_GAC_0060

The coding of the SSM according to the status signal for a Discrete type label is done according to the table below (the "`type`" flag of the SSM parameter shall be equal to `status_ssm_dis)`:

#EndText
#Verify Test

Bit 31	Bit 30	Designation	Simulated value	Meaning
0	0	Normal Operation (N.O.)	0	The data is valid.
0	1	Not Computed Data (N.C.D.)	4	The emitter does not guarantee the data.
1	0	Functional Test (F.T.)	16	The data does not represent a valid value for the aircraft.
1	1	Failure Warning (F.W.)	8	The data is invalid (emitter invalid).

Table 13: Correspondence of SSM of a DIS label

E_STL_FDEF_A429_GAC_0070

The coding of the SSM according to the status signal for a BCD type label is done according to the table below (the "`type`" flag of the SSM parameter shall be equal to `status_ssm_bcd`)::

#EndText
#Verify Test

Bit 31	Bit 30	Designation	Simulated value	Meaning
0	0	Positive data	0	Plus, North, East, Right, To, Above
0	1	Not Computed Data (N.C.D.)	4	The emitter does not guarantee the data.
1	0	Functional Test (F.T.)	16	The data does not represent a valid value for the aircraft.
1	1	Negative data	0	Minus, South, West, Left, From Below

Table 14: Correspondence of SSM of a BCD label

E_STL_FDEF_A429_GAC_0080

The coding of the SSM for an ISO5 type label is done in the same way as for a BNR label if the "`type`" flag of the SSM parameter is equal to `status_ssm_bnr` or in the same way as a BCD label if the "`type`" flag of the SSM parameter is equal to `status_ssm_bcd`. The SSM is not coded if the "`type`" flag of the SSM parameter is equal to `status_no_ssm`. In this last case, a warning shall be produced.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0085

The coding of the SSM for an HYB type label is done in the same way as for a BNR label if the "`type`" flag of the SSM parameter is equal to `status_ssm_bnr` or in the same way as a Discrete label if the "`type`" flag of the SSM parameter is equal to `status_ssm_dis`. The

SSM is not coded if the "`type`" flag of the SSM parameter is equal to `status_no_ssm`. In this last case, a warning shall be produced.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0090

The coding of the SSM for an Opaque type label is done in the same way as ISO5 labels.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0095

The FDEF GAC produces a variable of Boolean type for the Refresh status of the A429 label the name of which is constructed from the "name" field of the "A429Label" flag of the xml: <A429Label name>_R files.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0096

The value of the refresh Boolean on formatting depends on the conditions on the status signal and SSM of the A429 label:

#EndText
#Verify Test

Refresh value	Condition
1	The value of the status signal of the A429 label is not 2 (Not Emitted).
0	The value of the status signal of the A429 label is 2 (Not Emitted).

Table 15: Condition on Refresh value on A429 formatting

E_STL_FDEF_A429_GAC_0097

The values of the status signals on deformatting depend on the conditions on the refresh Booleans of the A429 label:

#EndText
#Verify Test

Refresh value	Condition
1	The status signal is converted according to the SSM value if the refresh Boolean of the A429 label is equal to 1.
0	The value of the status signal of the A429 label is set to 2 (Not Emitted).

Table 16: Condition on status values on A429 deformatting

E_STL_FDEF_A429_GAC_0100

If the validity of the ARINC A429 label does not exist in the configuration tables, the ARINC data is valid by default.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0110

Signals of Boolean type coded in an ARINC A429 label are defined as follows:

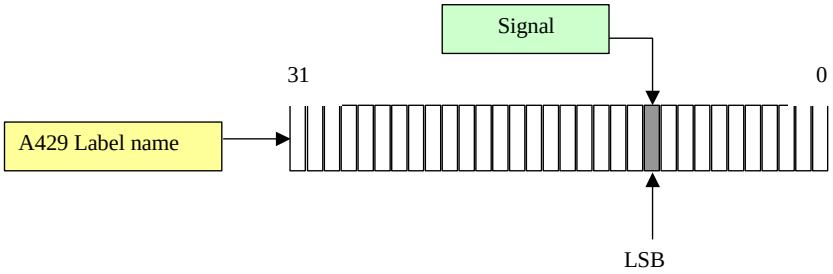

Figure 17: Location of Boolean signal in an A429 label

The LSB value is given by the "lsb" field.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0120

The signals of float type are coded in ARINC A429 label as follows:

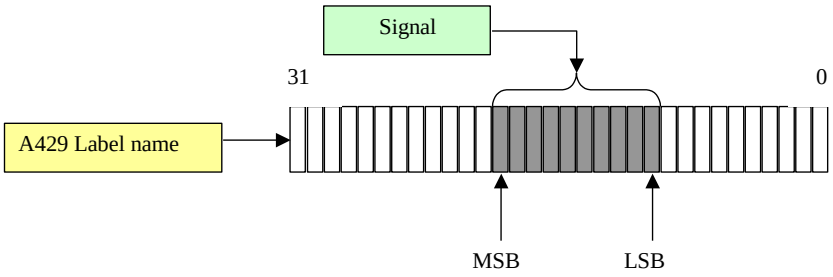

Figure 18: Location of float signal in an A429 label

The LSB value is given by the "lsb" field, the number of bits assigned by the "nbBit" field and the resolution by the "floatResolution" field of the "signal" flag.

The coding of the float depends on the type of label. Coding can be of "BNR" type or "BCD" type.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0130

Signals of integer type are coded in ARINC A429 label as follows:

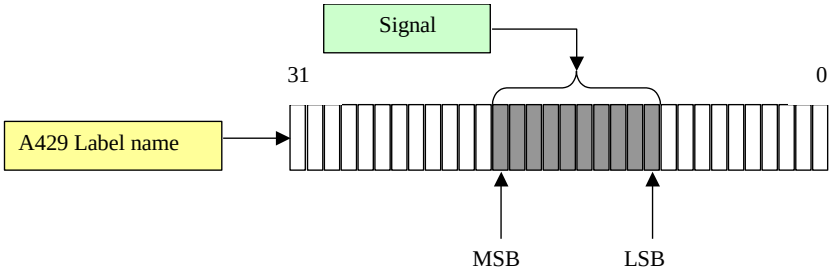

Figure 19: Location of integer signal in an A429 label

The LSB value is given by the "lsb" field, the number of bits assigned by the "nbBit" field and the resolution by the "integerResolution" field of the "signal" flag.

The coding of the float depends on the type of label. Coding can be of "BNR" type or "BCD" type.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0140

Signals of "Opaque" and "String" type coded in an ARINC A429 label are defined as follows:

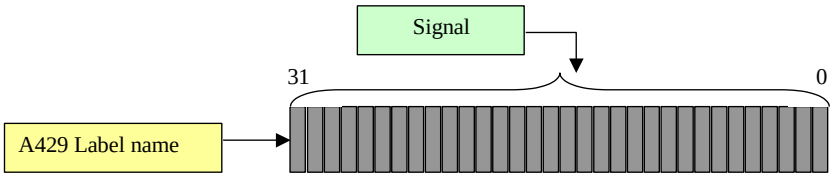

Figure 20: Location of opaque or string signal in an A429 label

The value of the signal is copied with no transformation or formatting.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0150

The specific case of labels of "ISO5" type are coded as follows in ARINC A429 label:

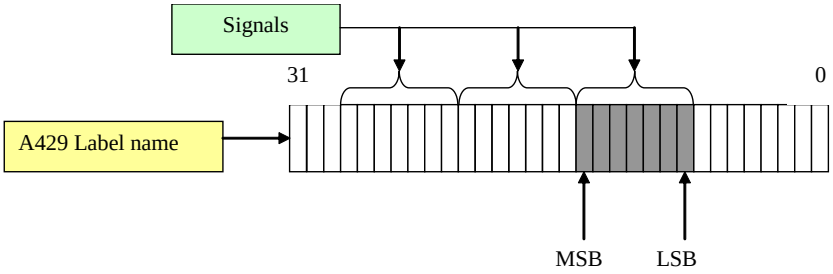

Figure 21: Location of char signal in an A429 ISO5 label

The LSB value is given by the "lsb" field, the number of bits assigned by the "nbBit" field of the "signal" flag. Each signal of string type is coded over 7 bits in the ARINC A429 label from the lsb.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0160

The A429 preformatted variables are initialised if the "init" field of the "signal" flag is completed by a coherent value.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0165

For char array signals, the “init” field can be filled with several forms as illustrated in the following example.

The string ABBCCC can be written as:
- ABBCCC
- A/B/B/C/C/C
- ‘A’/’B’/’B’/’C’/’C’/’C’
- A/B*2/C*3
- ‘A’/’B’*2/’C’*3

The “/” character shall be escaped with a “\” if it is not a delimiter.

#EndText
#Verify Test

E_STL_FDEF_A429_GAC_0170

The FDEF GAC applies a conversion function given by the "convention" field of the "signal" flag during the formatting of a signal or the deformatting of an A429 label if it is not empty.

#EndText
#Verify Test

5.3.4 Processing relevant to CAN messages

In the remainder of this document, the LSB is numbered 0, the MSB is numbered 63. The following requirements describe that which the generated code must ensure for the formatting or deformatting of CAN messages.

E_STL_FDEF_CAN_GAC_0010

The code generated by GAC must ensure the following conversions for the CAN messages:

- CAN message <–> float
- CAN message <–> integer
- CAN message <–> Boolean
- CAN message <–> character string
- CAN message <–> opaque

In the generated code, the Boolean type shall be an integer the value 0 of which shall correspond to FALSE and the other values to TRUE.

#EndText
#Verify Test

E_STL_FDEF_CAN_GAC_0020

The name of the formatted CAN variable is given by the "name" attribute of the "message" flag of the `CAN_prod.xml` and `CAN_conso.xml` files. The type of variable is a 64-bit integer:

- "long" on an OSF1 station (with 64-bit compiler)
- "long" on a SunOS station (with 64-bit compiler)
- "__int64" on a Windows 2000 PC
- "__int64" on a Linux PC

With an option on command line, a table with two 32-bit integers can be used instead of a single 64-bit integer.

The name of a preformatted variable is given by the "name" attribute of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_CAN_GAC_0025

The FDEF GAC produces a Boolean type variable for the Refresh status of the CAN message the name of which is constructed from the "name" field of the "message" flag of the xml: <message name>_R files.

#EndText
#Verify Test

E_STL_FDEF_CAN_GAC_0026

The value of the refresh Boolean on formatting depends on the conditions on the status signal of the CAN message:

#EndText
#Verify Test

Refresh value	Condition
1	The value of the status signal of the CAN message is not 2 (Not Emitted).
0	The value of the status signal of the CAN message is 2 (Not Emitted).

Table 17: Condition on Refresh value on CAN formatting

E_STL_FDEF_CAN_GAC_0027

The values of the status signals on deformatting depend on the conditions on the refresh Boolean of the CAN message:

#EndText
#Verify Test

Refresh value	Condition
1	The status signal is set to value 0 (Normal Operation) if the refresh Boolean of the CAN message is equal to 1.
0	The value of the status signal of the CAN message is set to 2 (Not Emitted).

Table 18: Condition on status values on CAN deformatting

E_STL_FDEF_CAN_GAC_0030

The Booleans coded in a CAN message are defined as follows:

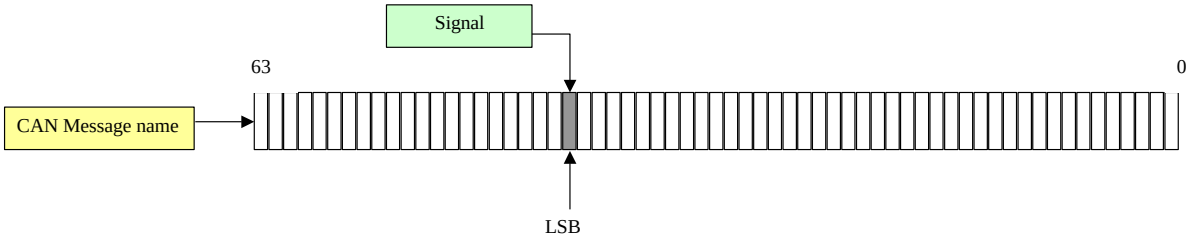

Figure 22: Location of Boolean signal in a CAN message

#EndText
#Verify Test

E_STL_FDEF_CAN_GAC_0040

Floats and integers coded in a CAN message are defined as follows:

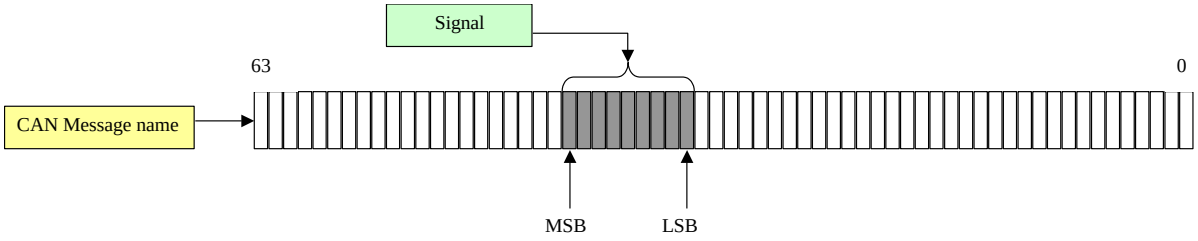

Figure 23: Location of float and integer signal in a CAN message

Coding can be of "BNR" type, "BCD" type or “BN2” type for integers and “Simple Precision” or “BNR” for floats.

#EndText
#Verify Test

E_STL_FDEF_CAN_GAC_0050

Signals of "Opaque" and "String" type coded in a CAN message are defined as follows:

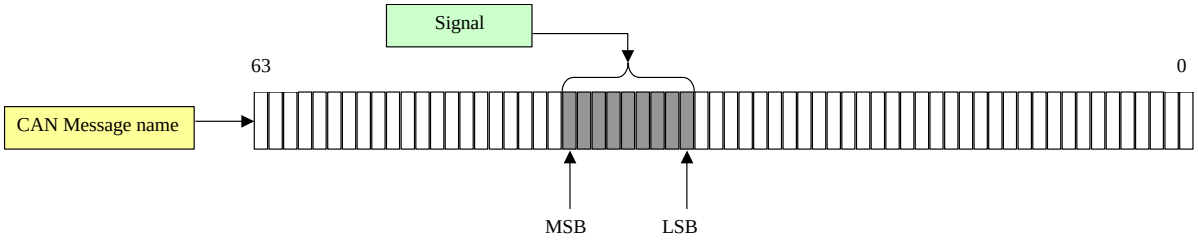

Figure 24: Location of opaque or string signal in a CAN message

The value of the signal is copied with no transformation or formatting.

#EndText
#Verify Test

E_STL_FDEF_CAN_GAC_0060

The preformatted CAN variables are initialised if the "init" field of the "signal" flag are completed by coherent values.

#EndText
#Verify Test

E_STL_FDEF_CAN_GAC_0065

For char array signals, the “init” field can be filled with several forms as illustrated in the following example.

The string ABBCCC can be written as:
- ABBCCC
- A/B/B/C/C/C
- ‘A’/’B’/’B’/’C’/’C’/’C’
- A/B*2/C*3
- ‘A’/’B’*2/’C’*3

The “/” character shall be escaped with a “\” if it is not a delimiter.

#EndText
#Verify Test

E_STL_FDEF_CAN_GAC_0070

The FDEF GAC applies a conversion function given in the "convention" field of the "signal" flag during the formatting of a signal or the deformatting of a CAN message if it is not empty.

#EndText
#Verify Test

5.3.5 Processing relevant to MIL1553 messages

In the remainder of this document, the LSB is numbered 0, the MSB is represented by character "n" corresponding to the length of the MIL1553 message. The following requirements describe that which the generated code must ensure for the formatting or deformatting of MIL1553 messages.

E_STL_FDEF_M1553_GAC_0010

The code generated by GAC must ensure the following conversions for the MIL1553 messages:

- MIL1553 payload <–> float
- MIL1553 payload <–> integer
- MIL1553 payload <–> Boolean
- MIL1553 payload <–> character string
- MIL1553 payload <–> opaque

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0020

In the generated code, the Boolean type shall be an integer the value 0 of which shall correspond to FALSE and the other values to TRUE.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0030

The name of the formatted MIL1553 message is given by the "name" attribute" of the "message" flag of the `M1553_prod.xml` and `M1553_conso.xml` files.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0040

The name of a preformatted variable called signal is given by the "name" attribute of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0050

The signals are formatted over a 16-bit word and consist of a "Functional Data Word". A "Functional Data Word" contains only one type of signal.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0060

For non-Boolean signals, a "Functional Data Word" contains only one signal.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0070

The type of MIL1553 payload is a char table the size of which is given by the "length" attribute of the "message" flag. The size in the xml file is given in number of 16-bit words.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0080

A MIL1553 payload consists of one or more 16-bit "Functional Data Words".

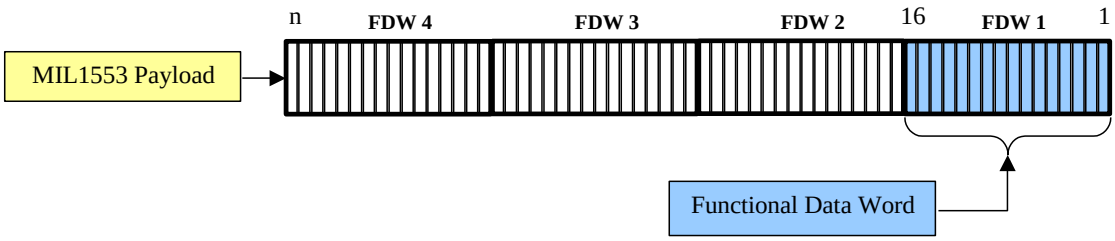

Figure 25: Description of a Functional Data Word

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0090

The FDEF GAC produces a variable of Boolean type for the Refresh status of the MIL1553 message the name of which is constructed from the "name" field of the "message" flag of the xml: <message name>_R files.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0100

The coding of the FDW data according to the status signal is done according to the table below:

#EndText
#Verify Test

Value of formatted data	Designation	Simulated value	Meaning
Formatted data	Normal Operation (N.O.)	0	The data is valid.
Bits 1 to 15 at 0 Bit 16 at 1	Not Normal Operation (N.N.O.)	4	The emitter does not guarantee the data.
Formatted data	Not Emitted (N.E.)	2	The data is not emitted.

Table 19: Correspondence of status signal for an FDW

E_STL_FDEF_M1553_GAC_0110

The coding of the refresh and of the data of the MIL1553 message according to the status signal for all types of MIL1553 messages is done according to the table below:

#EndText
#Verify Test

Refresh value	Input parameter	Condition
1	N/A	No status signal of the FDWs of the MIL1553 payload is set to Not Emitted (value 2).
0	/REFRESH_LOG0	At least one status signal of the FDWs of the MIL1553 payload is set to Not Emitted (value 2).
0	/REFRESH_LOG1	All the status signals of the FDWs of the MIL1553 payload are set to Not Emitted (value 2).

Table 20: Refresh condition during formatting of a MIL1553 payload

E_STL_FDEF_M1553_GAC_0120

The coding of the refresh and of the data of the MIL1553 message according to the status signal for all types of MIL1553 messages is done according to the table below:

#EndText
#Verify Test

Refresh value	Condition
1	No status signal of an FDW of the MIL1553 payload is set to Not Emitted (value 2).
0	All the status signals of the FDWs of the MIL1553 payload are set to Not Emitted (value 2).

Table 21: Status condition during deformatting of a MIL1553 payload

E_STL_FDEF_M1553_GAC_0130

Signals of Boolean type are coded in the MIL1553 payload on a bit of the Functional Data Word:

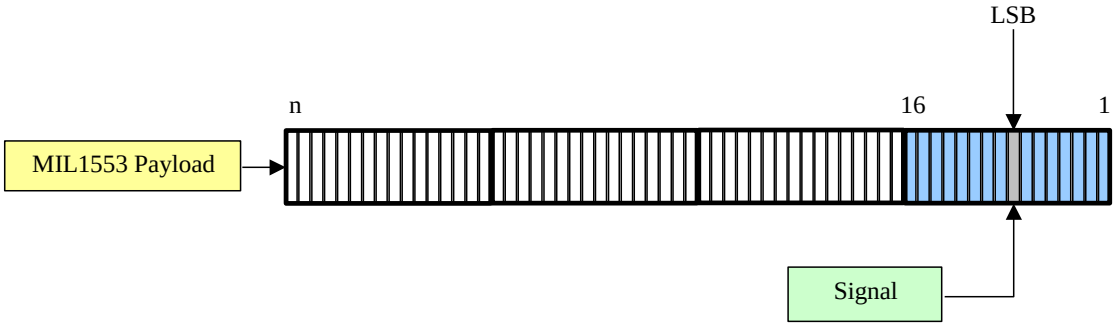

Figure 26: Coding of Boolean signal in a MIL1553 message

The LSB value is given by the "lsb" field of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0140

Signals of float type are coded in the MIL1553 message over a number of bits varying between 1 and 16:

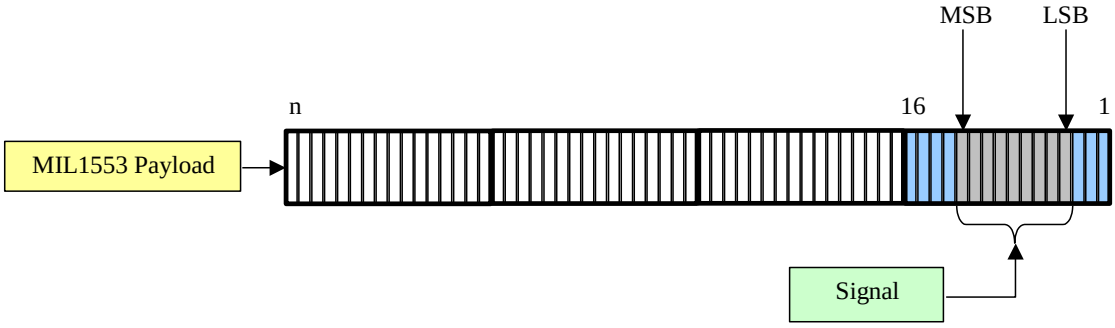

Figure 27: Coding of float signal in a MIL1553 message

The LSB value is given by the "lsb" field, the number of bits assigned by the "nbBit" field and the resolution by the "floatResolution" field of the "signal" flag.

The coding of the float is of twos complement type called "BNR" given by the "floatCodingType" field of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0150

Signals of integer type are coded in the MIL1553 message over a number of bits varying between 1 and 16:

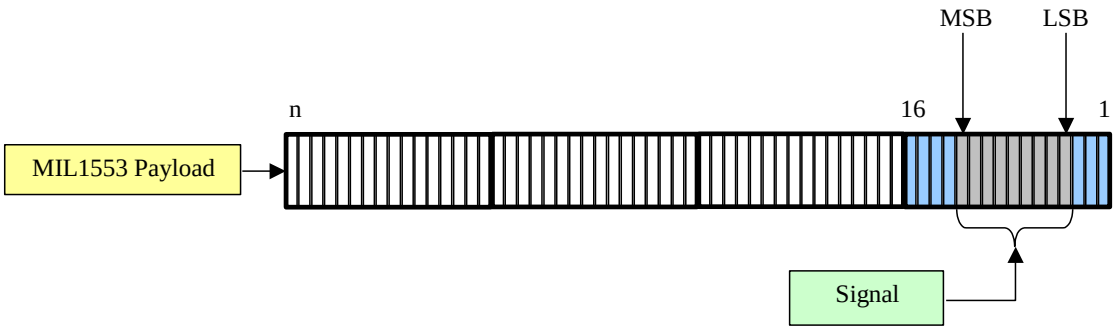

Figure 28: Coding of integer signal in a MIL1553 message

The LSB value is given by the "lsb" field, the number of bits assigned by the "nbBit" field and the resolution by the "integerResolution" field of the "signal" flag.

The coding of the integer signal is of twos complement type called "BNR" given by the "integerCodingType" field of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0160

Signals of string type (character string) are coded in the MIL1553 message over a number of bits varying between 1 and 16:

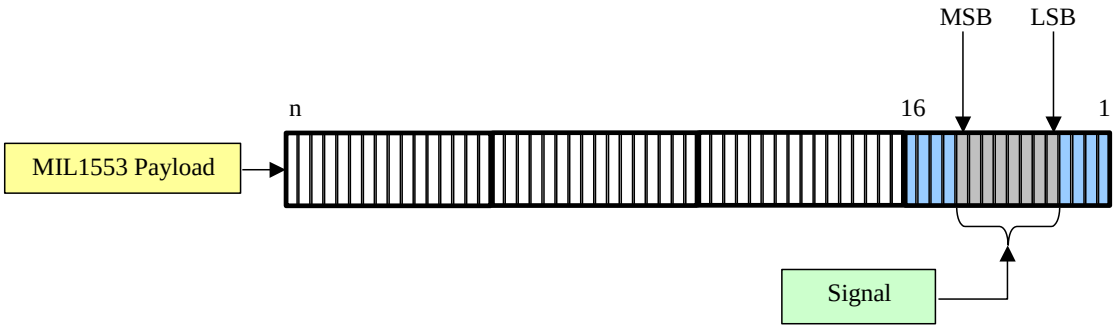

Figure 29: Coding of string signal in a MIL1553 message

The GAC is in charge of directly copying the number of bytes of the character string into the Functional Data Word part of the MIL1553 message. The LSB value is given by the "lsb" field, the MSB value by the "msb" field and the number of bits assigned by the "nbBit" field of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0170

Signals of opaque type (over 2 bytes) are coded in the MIL1553 message over a number of bits varying between 1 and 16:

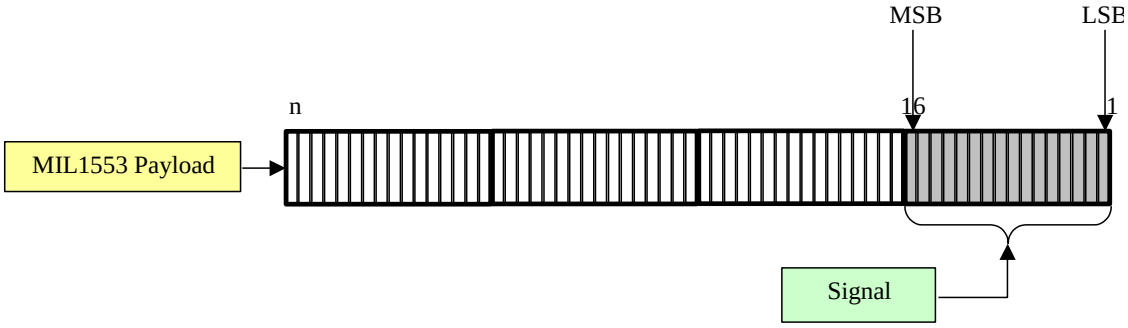

Figure 30: Coding of opaque signal in a MIL1553 message

The GAC is in charge of copying the complete two bytes of the opaque signal into the Functional Data Word part of the MIL1553 message. The LSB value is given by the "lsb" field, the MSB value by the "msb" field and the number of bits by the "nbBit" field of the "signal" flag.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0180

The preformatted M1553 variables are initialised if the "init" field of the "signal" flag is completed by a coherent value.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0185

For char array signals, the “init” field can be filled with several forms as illustrated in the following example.

The string ABBCCC can be written as:
- ABBCCC
- A/B/B/C/C/C
- ‘A’/’B’/’B’/’C’/’C’/’C’
- A/B*2/C*3
- ‘A’/’B’*2/’C’*3

The “/” character shall be escaped with a “\” if it is not a delimiter.

#EndText
#Verify Test

E_STL_FDEF_M1553_GAC_0190

The FDEF GAC applies a conversion function given by the "convention" field of the "signal" flag during the formatting of a signal or the deformatting of a MIL1553 payload if it is not empty.

#EndText
#Verify Test

5.3.6 Generic processing

E_STL_FDEF_GENP_GAC_0010

A data type may differ between the AICD and the MICD for a given preformatted port. GAC shall implement the following conversion rules:

AICD type	MICD type	Action
Boolean	any type	the Boolean type is forced, a warning shall be displayed if the MICD type was different
Integer/Enumerate	bool /char	unsupported, a warning shall be displayed
	int	Conversion not needed
	float/double	the conversion shall be done with a cast
Float	bool/char	unsupported, a warning shall be displayed
	int/double	the conversion shall be done with a cast
	float	Conversion not needed
String/Opaque	any type	the char type is forced, a warning shall be displayed if the MICD type was different

Table 22: Conversion rules

#EndText
#Verify Test

E_STL_FDEF_GENP_GAC_0020

If a conversion fails the formatting/deformating code shall not be produced.

#EndText
#Verify Test

E_STL_FDEF_GENP_GAC_0030

If a conversion fails the initialisation code shall not be produced.

#EndText
#Verify Test

5.3.7 Files produced by GAC

E_STL_FDEF_PROD_GAC_0010

The data to be produced or to be consumed (preformatted data, ARINC words, AFDX payloads, CAN messages, MIL1553 payloads) must be defined in a specific file.

#EndText
#Verify Test

E_STL_FDEF_PROD_GAC_0020

The generated code must have three main input points each corresponding to a first-level function:

- "Initialisation" which contains the code initialising all the elementary variables and the data of the generated code with a default value.

- "Decoder" which contains the code which deformats the AFDX payloads, the ARINC 429 labels, the CAN messages and/or the MIL1553 payloads into preformatted data.

- "Encoder" which contains the code which formats the preformatted data into AFDX payloads, ARINC 429 labels, CAN messages and/or MIL1553 payloads.

#EndText
#Verify Test

E_STL_FDEF_PROD_GAC_0030

If the /MODEL_ID option is not used, GAC produces five files concerning the AFDX payloads, ARINC A429 labels and CAN messages:

- `EYY_format.c`: C code containing the formatting of the preformatted variables into AFDX payloads, ARINC A429 label, CAN message and/or MIL1553 payloads,

- `EYY_deformat.c`: C code containing the deformatting of the AFDX payloads, ARINC A429 label, CAN message and/or MIL1553 payloads into preformatted variables,

- `EYY_initialisation.c`: C code containing the FS conversion functions of the AFDX payloads, the ARINC SSM and the initialisation function of the preformatted or formatted variables associated with an initialisation value,

- `EYY_initialisation.h`: C code containing the definitions of the conversion and initialisation functions developed in the `EYY_initialisation.c` file,

- `EYY_stdAfx.h`: C code containing the definitions of the masks and macros used in the code,

- `EYY_var.h`: declarations of formatted and deformatted variables (produced with the /INTEGRE command option),

- `trame.dec`: declarations of formatted and deformatted variables in .dec file format (produced with /AVEC_MODELE command option).

#EndText
#Verify Test

E_STL_FDEF_PROD_GAC_0031

If the /MODEL_ID <modelId> option is used, GAC produces five files concerning the AFDX payloads, ARINC A429 label and CAN messages:

- `modelId_format.c`: C code containing the formatting of the preformatted variables into AFDX payloads, ARINC A429 label, CAN message and/or MIL1553 payloads,

- `modelId_deformat.c`: C code containing the deformatting of the AFDX payloads, ARINC A429 label, CAN message and/or MIL1553 payloads into preformatted variables,

- `modelId_initialisation.c`: C code containing the FS conversion functions of the AFDX payloads, the ARINC SSM and the initialisation function of the preformatted or formatted variables associated with an initialisation value,

- `modelId_initialisation.h`: C code containing the definitions of the conversion and initialisation functions developed in the modelId`_initialisation.c` file,

- `modelId_stdAfx.h`: C code containing the definitions of the masks and macros used in the code,

- `modelId_var.h`: declarations of formatted and deformatted variables (produced with the /INTEGRE command option),

- `modelId.dec`: declarations of formatted and deformatted variables in .dec file format (produced with the /AVEC_MODELE command option).

#EndText
#Verify Test

E_STL_FDEF_PROD_GAC_0040

FDEF produces a log file, `FDEF.log`, to trace all the code generation problems.

#EndText
#Verify Test

5.3.8 Operating modes

E_STL_FDEF_OP_GAC_0010

Three operating modes are identified for the code generator:

- Nominal mode, the generator can generate the formatting/deformatting code without problems.

- Degraded mode: the generator can partially generate the formatting/deformatting code; however anomalies have been detected.

- Error mode: the generator cannot generate the code, even partially.

#EndText
#Verify Test

E_STL_FDEF_OP_GAC_0020

The FDEF tool must go to error mode in the following cases:

- syntax error in the XML files,

- incoherence in the structure of the XML files.

#EndText

#Verify Test

E_STL_FDEF_OP_GAC_0030

Any anomalies recorded on the generation of the code must lead to a trace in the `FDEF.log` file, specifying the line causing the problem in the XML files.

#EndText
#Verify Test

E_STL_FDEF_OP_GAC_0040

The FDEF tool must go to degraded mode in the following cases:

- incomplete description of parameter in the XML file.

#EndText
#Verify Test

5.3.9 First-level function

There are three first-level functions:

- "Load the Configuration tables" which loads the information contained in the XML files at FDEF input and creates in memory the data trees corresponding to this information and which will be consulted by the formatting and deformatting code generation functions.
- "Generate the formatting and deformatting code".
- "Delete the data trees" which deletes the data trees loaded during the reading of the XML files to free allocated memory space.

E_STL_FDEF_FONC_GAC_0010

All of the XML configuration tables must be read by the table loading function.

#EndText
#Verify Test

E_STL_FDEF_FONC_GAC_0020

If a configuration table is absent, the table loading function must formulate a warning and consider this case as being identical to the case of an empty table.

#EndText
#Verify Test

E_STL_FDEF_FONC_GAC_0030

Preformatted data can be transported by several AFDX messages.

#EndText

#Verify Test

E_STL_FDEF_FONC_GAC_0040

Conversion between the FS and the data validity variable (SSM, Mask, etc.) must be parameterisable via call of conversion functions (conversion flag of the XML file).

#EndText
#Verify Test

5.3.10 GAC command

E_STL_FDEF_CMD_GAC_0010

The command line for the execution of the GAC includes the following generation options.

The command line format is as follows:

```
fdef <options>
```

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0020

GAC accepts the following options concerning the SWAP of the AFDX payloads, CAN messages and MIL1553 payloads:

- `/SWAP`: the generated code swaps the AFDX payloads, the CAN messages and the MIL1553 payloads (by default),
- `/NO_SWAP`: the generated code swaps no data.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0030

GAC accepts the following options concerning the output format of the generated files:

- `/INTEGRE`: the generated code can be integrated into a simulation model (by default),
- `/AVEC_MODELE`: the generated code comprises a simulation model,
- `/AUTONOME`: the generated code comprises an autonomous model.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0040

GAC accepts the following options concerning the type of AFDX payload, ARINC A429 label or CAN message produced:

- /FORMAT: the generated code produces completely formatted AFDX payloads, ARINC A429 labels, CAN messages or MIL1553 payloads (by default),

- /PRE_FORMAT: the generated code produces preformatted AFDX payloads, ARINC A429 labels, CAN messages or MIL1553 payloads,

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0050

GAC accepts the following options concerning the AFDX modification words:

- /SS_MOT_MODIF: the generated code has no AFDX payload modification words (by default),

- /AV_MOT_MODIF: the generated code has AFDX payload modification words.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0060

GAC accepts the following options concerning the generated code input functions:

- /MONO_FONCTION: the generated code has only one main function for all the words to be formatted or deformatted (by default),

- /MULTI_FONCTION: the generated code has a function per word to be formatted or deformatted.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0070

GAC accepts the options concerning the type of data to be formatted or deformatted (not exclusive):

- /ALL: all types of words are taken into account (by default),

- /AFDX: AFDX payloads are taken into account,

- /A429: ARINC A429 labels are taken into account,

- /CAN: CAN messages are taken into account,

- /M1553: MIL1553 messages are taken into account.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0080

GAC accepts the following options concerning the type of CAN messages:

- /CAN_CHAR: the code generates CAN messages of char table type (by default),

- /CAN_LONG: the code generates CAN messages of long type,

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0085

GAC accepts the following options concerning the type of A429 and CAN messages:

- /TYPE_SIG: the generated code of A429 and CAN signed messages (by default),
- /TYPE_UNS: the generated code of A429 and CAN unsigned messages.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0090

GAC accepts the following options concerning the monitoring of the AFDX payloads:

- /SS_MONITEUR: the generated code does not take monitoring into account (by default),
- /MONITEUR_ATA42 <csv_file>: the generated code takes monitoring into account with the "csv_file" configuration file.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0100

GAC accepts the following options concerning the mnemo:

- /MNEMO <mnemo>: defines a mnemo or "mnemo" name.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0110

GAC accepts the following options concerning the refresh logic of the AFDX payloads and MIL1553 payloads:

- /REFRESH_LOG0: the generated code has a refresh logic such that if an FS is set to NE for the AFDX or the data is equal to 0x8000 for MIL1553, refresh is at 0, else at 1 (by default),
- /REFRESH_LOG1: the generated code has a refresh logic such that if all the FSs are set to NE for the AFDX or the data is equal to 0x8000 for MIL1553, refresh is at 0, else at 1.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0120

The /MONITEUR_ATA42 and /MNEMO options are complementary. If /MONITEUR_ATA42 is used, /MNEMO shall also be used.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0130

The /AVEC_MODELE and /MULTI_FONCTION options are exclusive.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0140

GAC accepts the /ALIC option to trigger the adding of the coding of the ALIC certificate to ports declared ALIC in the xml files at input.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0150

GAC accepts the /MODEL_ID <modelId> option to indicate that the generated .c, .h and .dec files are named according to the AMO naming rule (for the "modelId" model).

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0160

GAC accepts the / REFRESH_RATE_COEFF <coeff> option to indicate that a latency must be taken into account while generating the deformating code. A condition deduced from the coeff parameter and refresh data will allow the code to be executed.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0170

GAC accepts the /IGNORE_FORMAT_INIT option to indicate that initialisation code shall not be produced for ports that will be formatted.

#EndText
#Verify Test

E_STL_FDEF_CMD_GAC_0180

GAC accepts the /IGNORE_DEFORMAT_INIT option to indicate that initialisation code shall not be produced for ports resulting from the deformatting process.

#EndText
#Verify Test

5.4 ADCN-Simu function specification

This paragraph includes all the requirements relevant to the ADCN-Simu function.

5.4.1 ADCN-Simu function input files

E_STL_FDEF_ENT_ADCN_SIMU_0010

Code generation is done by analysing the following files:

- AC/ICD containing the description of all the model variables (name, type, length and position in message, etc.); several AC/ICDs can be taken into account.
- MD/ICD containing the names of the signals and the key making the link with the associated message in the AC/ICD of AFDX type.
- ADCN file containing the description of the ADCN network.

#EndText
#Verify Test

5.4.2 Processing relevant to AFDX payloads

E_STL_FDEF_AFDX_ADCN_SIMU_0010

ADCN-Simu processes the AFDX_INPUT_VL and AFDX_OUTPUT_VL lines of the AC/ICDs to get the identifier and the name of the AFDX payload.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ADCN_SIMU_0020

For each line, ADCN-Simu records the data contained in the following columns:

- `VL Identifier` (identifier included in the name of the AFDX payload)
- `VL name` (name associated with the AFDX payload)

#EndText
#Verify Test

E_STL_FDEF_AFDX_ADCN_SIMU_0030

ADCN-Simu processes the AFDX_INPUT_MESSAGE and AFDX_OUTPUT_MESSAGE lines of the AC/ICDs to get the list of AFDX payloads.

#EndText
#Verify Test

E_STL_FDEF_AFDX_ADCN_SIMU_0040

For each line, ADCN-Simu records the data contained in the following columns:

- `Message name` (included in the name of the AFDX payload)
- `Associated VL` (link to find the "`VL Identifier`" of an AFDX payload by comparing it with the "`VL name`")
- `Application name` (application associated with the payload)

- `FDS name` (name of FDS)
- `FS address in the message` (address of FS in bytes in AFDX payload)

#EndText
#Verify Test

E_STL_FDEF_AFDX_ADCN_SIMU_0050

For an `Input Refresh` the unique key which differentiates between two refreshs is given by:

```
[Application_Name]:[Associated VL]:[Message Name]
```

#EndText
#Verify Test

E_STL_FDEF_AFDX_ADCN_SIMU_0051

For an `Input Status` the unique key which differentiates between two status is given by:

```
[Application_Name]:[Associated VL]:[Message Name]:[FDS Name]
```

#EndText
#Verify Test

E_STL_FDEF_AFDX_ADCN_SIMU_0052

For an `Output Refresh` the unique key which differentiates between two refreshs is given by:

```
[Application_Name]:[Message Name]
```

#EndText
#Verify Test

E_STL_FDEF_AFDX_ADCN_SIMU_0053

For an `Output Status` the unique key which differentiates between two status is given by:

```
[Application_Name]:[Message Name]:[FDS Name]
```

#EndText
#Verify Test

E_STL_FDEF_AFDX_ADCN_SIMU_0060

The types of signals allowed are:

- Boolean for a `Refresh`
- Integer for a `Status`

ADCN-Simu returns a warning for all signals of other types.

#EndText
#Verify Test

5.4.3 Processing of MD/ICD

E_STL_FDEF_MD_ADCN_SIMU_0010

ADCN-Simu processes the lines of the MD/ICD with a `Status` column equal to `True` to get the list of formatted variables of the model. The MD/ICD passed by parameters is related to the AC/ICD INPUTS.

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_SIMU_0020

For each variable, ADCN-Simu records the data contained in the following columns of the MD/ICD:

- `Name` (name of signal)
- `Type` (type of signal)
- `Com Format` (signal communication format)
- `Aircraft Signal Name` (joint key to relate the signal to the AC/ICD)
- `Status` (refresh signal or not)
- `Interface Level`

#EndText

#Verify Test

E_STL_FDEF_MD_ADCN_SIMU_0030

For each variable, ADCN-Simu creates a `Type Port` which is equal to:

- `Refresh` (if `Interface Level` column equal to `Format`)
- `Status` (if `Interface Level` column equal to `Deformat`)

#EndText

#Verify Test

E_STL_FDEF_MD_ADCN_SIMU_0040

The unique key which differentiates between two variables within a given model is given by the "`Name`" column.

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_SIMU_0050

Each variable of an MD/ICD is related to the AC/ICDs by means of the unique key entered into the "`Aircraft Signal Name`" column.

#EndText
#Verify Test

5.4.4 Processing of ADCN

E_STL_FDEF_ADCN_ADCN_SIMU_0010

ADCN-Simu processes the lines of the ADCN file to get the list of validity equations.

#EndText
#Verify Test

E_STL_FDEF_ADCN_ADCN_SIMU_0020

For each line, ADCN-Simu records data contained in the following columns of the ADCN file:

- `Model Name` (name of the model)
- `Application Name` (application associated with the message)
- `VL Identifier` (identifier included in the name of the message)
- `Message Name` (name of the message)
- `Validity Equation`

#EndText

#Verify Test

E_STL_FDEF_ADCN_ADCN_SIMU_0030

The unique key which differentiates between two validity equations is given by:

```
[Model Name];[Application Name];[VL Identifier];[Message Name]
```

#EndText
#Verify Test

5.4.5 Files produced by ADCN-Simu

E_STL_FDEF_PROD_ADCN_SIMU_0010

The generated code must have two main input points each corresponding to a first-level function:

- "Initialisation" which contains the code that initialises all the elementary variables and the data of the generated code with a default value.
- "Simul" which contains the code that simulates the ADCN network.

#EndText
#Verify Test

E_STL_FDEF_PROD_ADCN_SIMU_0020

With the required -–model_id=<modelId> option, ADCN-Simu produces six files concerning the AFDX payloads:

- `modelId_ADCN_simul.c`: C file containing the code that simulates the ADCN network,
- `modelId_ADCN_initialisation.c`: C code containing the FS conversion functions of the AFDX payloads and the initialisation function of the variables associated with an initialisation value,
- `modelId_ADCN_initialisation.h`: C code containing the definitions of the conversion and initialisation functions developed in the modelId`_initialisation.c` file,
- `modelId_ADCN_stdAfx.h`: C code containing the definitions of the masks and macros used in the code,
- `modelId_ADCN_var.h`: declarations of the variables.
- modelId_ADCN_switch_micd.csv : MICD file declaring ADCN network switch variables.

#EndText
#Verify Test

E_STL_FDEF_PROD_ADCN_SIMU_0030

The generated code must respect the following rules:

- AFDX messages must be sorted alphabetically
- In each AFDX message :
  - Refresh must be sorted alphabetically
  - Status must be sorted by FDS and then by FS

#EndText
#Verify Test

E_STL_FDEF_PROD_ADCN_SIMU_0040

For all the refreshs retrieved from the MICD, ADCN-Simu shall produce the following code:

```
<validity_variable> = <validity_equation>;
<refresh> = <refresh> && <validity_variable>;
```

With :

- `<validity_variable>` an int variable declared by ADCN-Simu
- `<validity_equation>`the validity equation retrieved from the ADCN file for this refresh
- `<refresh>` the refresh retrieved from the MICD.

#EndText
#Verify Test

E_STL_FDEF_PROD_ADCN_SIMU_0050

For all the status retrieved from the MICD, ADCN-Simu shall produce the following code without the –target_p option:

```
<validity_variable> = <validity_equation>;
If (!<refresh> || !<validity_variable>)
{
  <status> = NE;
}
else
{
  status = convertToStatus(<AFDX_message>[<FS_address]);
}
```

With :

- `<validity_variable>` an int variable declared by ADCN-Simu
- `<validity_equation>`the validity equation retrieved from the ADCN file for this refresh
- `<status>` the status retrieved from the MICD.
- `<AFDX_message>` the AFDX message in which the status is encoded.
- `<refresh>` the refresh corresponding to the AFDX message.
- `<FS_address>` the address in the AFDX message of the FS corresponding to the status.

#EndText
#Verify Test

E_STL_FDEF_PROD_ADCN_SIMU_0060

For all the status retrieved from the MICD, ADCN-Simu shall produce the following code without the –target_p option:

```
<validity_variable> = <validity_equation>;
If (!<validity_variable>)
{
  <status> = NE;
}
```

With :

- `<validity_variable>` an int variable declared by ADCN-Simu
- `<validity_equation>`the validity equation retrieved from the ADCN file for this refresh
- `<status>` the status retrieved from the MICD.

#EndText
#Verify Test

5.4.6 ADCN-Simu command

E_STL_FDEF_CMD_ADCN_SIMU_0010

The command line for the execution of the ADCN-Simu includes the following required options.

The command line format is as follows:

```
ADCN_Simu.py <icd2xml.ini> <aicd_list> <micd_file> <adcn_file>
--mnemo=<mnemo> --model_id=<modelId> [--target_p] [--not_standalone]
[--fdef_names] [--define_switch]
```

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_SIMU_0020

ADCN-Simu accepts the following options concerning the mnemo:

- `--mnemo=<mnemo>`: defines a mnemo or "mnemo" name.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_SIMU_0030

ADCN-Simu accepts the `--model_id=<modelId>` option to indicate that the generated .c and .h files are named according to the AMO naming rule (for the "modelId" model).

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_SIMU_0040

ADCN-Simu accepts the `-target_p` option to indicate that the generated code must process status directly without converting it from a SSM

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_SIMU_0050

ADCN-Simu accepts the `-not_standalone` option to indicate that the generated code will be used with code generated by GAC. Therefore the files: `modelId_ADCN_initialisation.c`, `modelId_ADCN_initialisation.h` and `modelId_ADCN_stdAfx.h` will not be generated, the `modelId_ADCN_var.h` file will declare only the validity equation variables and switch state variables, and the `modelId_ADCN_simul.c` will include the following header files:

- `modelId_stdAfx.h` instead of `modelId_ADCN_stdAfx.h`
- `modelId_initialisation.h` instead of `modelId_ADCN_initialisation.h`
- `modelId_var.h` in addition of `modelId_ADCN_var.h`

If this option is used, target_p options shall be forced.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_SIMU_0060

ADCN-Simu accepts the `--fdef_names` option to indicate that the generated code will create output ports to apply ADCN treatments instead of directly modify the input ports.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_SIMU_0070

ADCN-Simu accepts the `-define_switch` option to indicate that ADCN network switch variables shall be declared in the modelId_ADCN_var.h file instead of being declared in the MICD file. This MICD file shall not be generated in this case.

#EndText
#Verify Test

5.5 ADCN-FDEF function specification

This paragraph includes all the requirements relevant to the ADCN-FDEF function.

5.5.1 ADCN-FDEF function input files

E_STL_FDEF_ENT_ADCN_FDEF_0010

Code generation is done by using the following files with other FDEF tools:

- FDEF.ini file to specify the path of the subtools to use and their options
- AC/ICD containing the description of all the model variables (name, type, length and position in message, etc.); several AC/ICDs can be taken into account.
- MD/ICD containing the names of the signals and the key making the link with the associated message in the AC/ICD of AFDX type.
- ADCN file containing the description of the ADCN network; this file can be of ncd or csv format.

#EndText
#Verify Test

E_STL_FDEF_ENT_ADCN_FDEF_0020

The FDEF.ini files shall contain the following fields:

- ICD2XML: to specify the path to the ICD2XML launcher/executable.

- ICD2XML_OPTIONS: to specify extra options to use with ICD2XML (this field is optional).
- GAC: to specify the path to the GAC launcher/executable.
- GAC_OPTIONS: to specify extra options to use with GAC (this field is optional).
- ADCN-Simu: to specify the path to the ADCN-Simu launcher/executable.

#EndText
#Verify Test

5.5.2 Processing of MD/ICD

E_STL_FDEF_MD_ADCN_FDEF_0010

ADCN-FDEF extracts the FUN_IN and FUN_OUT sheets of a MICD of .xls type and creates to .csv file, fun_in.csv and fun_out.csv, which can be used by other FDEF sub-tools (ICD2XML or ADCN-Simu)

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_FDEF_0020

ADCN-FDEF extracts parts of a MICD of csv type according to the following parameters:

- Interface Level (‘Format’ or ‘Preformat’).
- Status (‘True’ or ‘False’).
- Com Format ('AFDX', 'ARINC', 'CAN', '1553', 'DIS' or 'ENV').

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_FDEF_0030

ADCN-FDEF can invert the linking key of a parameter by changing its Com Format to ENV and, if it s needed, by changing the name from ‘_I’ to ‘_O’ (with FDEF default naming rules).

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_FDEF_0040

ADCN-FDEF generates a new MICD file of .xls type by converting MICD files of .csv type which represents the FUN_IN and FUN_OUT sheets.

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_FDEF_0050

The parameters listed in the FUN_IN and FUN_OUT sheets of the final MICD are sorted by alphabetically by Com Format and the alphabetically by parameter name.

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_FDEF_0060

The name of the ouput MICD file of .xls type will be <model_id>.xls if –encapsulated option used or <model_id>_FDEF.xls otherwise. The <model_id> refers to the value of the option --model_id if it is used or the MOD parameter of the HEADER sheet of the input MICD file if the option is not used.

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_FDEF_0070

If the --encapsulated option is used, the following parameters of the HEADER sheet of the ouput MICD file will be modified:

-MOD=<model_id>_FDEF, with <model_id> the parameter described in E_STL_FDEF_MD_ADCN_FDEF_0060

-VER=<version>, with <version> the value of the --version option
-DAT=<date>, with <date> the current date
-REF=<version>, with version the current version of ADCN-FDEF tool

If the --encapsulated option is not used, only the DAT parameter is modified.
In both cases, all the other parameters will be copied from the input MICD.

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_FDEF_0080

If the --encapsulated option is not used, the PROFILE sheet of the final output MICD will be empty, else it will be a copy of the input MICD.

#EndText
#Verify Test

E_STL_FDEF_MD_ADCN_FDEF_0090

All the remaining sheets of the output MICD (sheets different of FUN_IN, FUN_OUT, HEADER and PROFILE) will be copied from the input MICD.

#EndText
#Verify Test

5.5.3 Process when --target=F

E_STL_FDEF_TF_ADCN_FDEF_0010

ADCN-FDEF launches ICD2XML with /INPUT_DEFORMAT and /MICD_NAMES options and then GAC with the /MODEL_ID option. The value of this option is defined by the --model_id option of ADCN-FDEF

#EndText
#Verify Test

E_STL_FDEF_TF_ADCN_FDEF_0020

If the --adcn option is activated, ADCN-FDEF launches ADCN-Simu with the same --mnemo and --model_id options as ADCN-FDEF and the --not_standalone, --target_p and –fdef_names option.

#EndText
#Verify Test

E_STL_FDEF_TF_ADCN_FDEF_0030

If the –encapsulated option is not activated; the final MICD file contains the following parts:

- In the FUN_IN sheet:
  - The AFDX refresh of the initial FUN_IN sheet if the --adcn option is used
  - The preformatted part of the initial FUN_OUT sheet with the inverted linking keys
  - The formatted equivalent of the preformatted part of the initial FUN_IN sheet
- In the FUN_OUT sheet:
  - The AFDX refresh of the initial FUN_IN sheet with the inverted linking keys if the --adcn option is used
  - The preformatted part of the initial FUN_IN sheet with the inverted linking keys
  - The formatted equivalent of the preformatted part of the initial FUN_OUT sheet

#EndText
#Verify Test

E_STL_FDEF_TF_ADCN_FDEF_0040

If the –encapsulated option is not activated; the final MICD file contains the following parts:

- In the FUN_IN sheet:
  - The formatted part of the initial FUN_IN sheet with the inverted linking keys
  - The formatted equivalent of the preformatted part of the initial FUN_IN sheet
- In the FUN_OUT sheet:

- The formatted part of the initial FUN_OUT sheet with the inverted linking keys
- The formatted equivalent of the preformatted part of the initial FUN_OUT sheet

#EndText
#Verify Test

E_STL_FDEF_TF_ADCN_FDEF_0050

The modelId_var.h file shall be renamed modelId_var_old.h file and a new modelId_var.h file shall be generated containing the following code:

```
#ifndef _INCLUDE_VAR_H
#define _INCLUDE_VAR_H

#ifndef logical
#define logical int
#endif

#ifndef False
#define False 0
#endif

#include "<modelId>_FDEF.h"
#endif
```

#EndText
#Verify Test

5.5.4 Process when --target=P

E_STL_FDEF_TP_ADCN_FDEF_0010

ADCN-FDEF launches ICD2XML with /INPUT_FORMAT and /A429_LABEL_NAME_RULE2 options and then GAC with the /MODEL_ID, /CAN_CHAR and /ALIC options. The value of the /MODEL_ID option is defined by the --model_id option of ADCN-FDEF

#EndText
#Verify Test

E_STL_FDEF_TP_ADCN_FDEF_0020

If the --adcn option is activated, ADCN-FDEF launches ADCN-Simu with the same --mnemo and --model_id options as ADCN-FDEF and the --not_standalone, --target_p and –fdef_names options.

#EndText
#Verify Test

E_STL_FDEF_TP_ADCN_FDEF_0030

If the –encapsulated option is not activated; the final MICD file contains the following parts:

- In the FUN_IN sheet:

- The AFDX refresh and status of the initial FUN_IN sheet if the --adcn option is used
- The formatted part of the initial FUN_OUT sheet with the inverted linking keys
- The preformatted equivalent of the formatted part of the initial FUN_IN sheet

- In the FUN_OUT sheet:
  - The AFDX refresh and status of the initial FUN_IN sheet with the inverted linking keys if the --adcn option is used
  - The formatted part of the initial FUN_IN sheet with the inverted linking keys
  - The preformatted equivalent of the formatted part of the initial FUN_OUT sheet

#EndText
#Verify Test

E_STL_FDEF_TP_ADCN_FDEF_0040

If the –encapsulated option is not activated; the final MICD file contains the following parts:

- In the FUN_IN sheet:
  - The preformatted part of the initial FUN_IN sheet with the inverted linking keys
  - The preformatted equivalent of the formatted part of the initial FUN_IN sheet
- In the FUN_OUT sheet:
  - The preformatted part of the initial FUN_OUT sheet with the inverted linking keys
  - The preformatted equivalent of the formatted part of the initial FUN_OUT sheet

#EndText
#Verify Test

E_STL_FDEF_TP_ADCN_FDEF_0050

The modelId_var.h file shall be renamed modelId_var_old.h file and a new modelId_var.h file shall be generated containing the following code:

```
#ifndef _INCLUDE_VAR_H
#define _INCLUDE_VAR_H

#ifndef logical
#define logical int
#endif
```

```
#ifndef False
#define False 0
#endif

#include "<modelId>_FDEF.h"
#endif
```

#EndText
#Verify Test

E_STL_FDEF_TP_ADCN_FDEF_0060

A file named modelId_dispatch_refresh.c shall be produced. This file shall contain a sp FDEF_dispatch_refresh function that link AFDX refresh generated by GAC with the one wich will be used by ADCn-Simu by a simple affectation. This shall be used in the modelId_main.c file.

#EndText
#Verify Test

5.5.5 Process when --target=N

E_STL_FDEF_TN_ADCN_FDEF_0010

The --adcn option must be used.

#EndText
#Verify Test

E_STL_FDEF_TN_ADCN_FDEF_0020

ADCN-FDEF launches ADCN-Simu with the same --mnemo and --model_id options as ADCN-FDEF and the –fdef_names option.

#EndText
#Verify Test

E_STL_FDEF_TN_ADCN_FDEF_0030

If the –encapsulated option is not activated; the final MICD file contains the following parts:

- In the FUN_IN sheet:
  - The AFDX refresh and status of the initial FUN_IN sheet
- In the FUN_OUT sheet:
  - The AFDX refresh and status of the initial FUN_IN sheet with the inverted linking keys

#EndText
#Verify Test

E_STL_FDEF_TN_ADCN_FDEF_0040

If the –encapsulated option is not activated; the final MICD file contains the following parts:

- The FUN_IN sheet is the same as the one in the initial MICD
- The FUN_OUT sheet is the same as the one in the initial MICD

#EndText
#Verify Test

5.5.6 Generation of a main file

E_STL_FDEF_MAIN_ADCN_FDEF_0010

ADCN-FDEF generated a main model c file that uses the generated code. It contains a <model_id>_FDEF_main method, with <model_id> the parameter defined in E_STL_FDEF_MD_ADCN_FDEF_0060. If the --encapsulated option is used, this method will call the <model_id> _main method of the input model.

#EndText
#Verify Test

5.5.7 ADCN-FDEF command

E_STL_FDEF_CMD_ADCN_FDEF_0010

The command line for the execution of the ADCN-FDEF includes the following required options.

The command line format is as follows:

```
adcn_fdef.py <fdef_ini> <aicds_path> <micd_file> [-m/--mnemo=<mnemo>]
[-i/--model_id=<modelId>] [-t/--target] [-e/--encapsulated]
[-n/--noswap] [-v/--version] [-a/--adcn] [<adcn_file>][-x/--xml_only]
[-g/--generate_dec] [--micd_m2633_2] [-f/--filt_network_B]
[-k/--keep_xml]
```

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0020

ADCN-FDEF accepts the following options concerning the mnemo:

- `-m/--mnemo=<mnemo>`: defines a mnemo or "mnemo" name.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0030

ADCN-FDEF accepts the `-i/--model_id=<model_id>` option to indicate that the generated .c and .h files are named according to the AMO naming rule (for the "modelId"

model). If this option is not used, the value of the MOD parameter of the HEADER sheet of the input MICD file will be used instead.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0040

ADCN-FDEF accepts the `-t/--target=<target>` option to specify which type of model is expected as output of the tool. Three values are accepted:

- 'F' for a formatted model
- 'P' for a preformatted model
- 'N' if the type of model is not specified; is this value is chosen, the –adcn option must be used.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0050

ADCN-FDEF accepts the `-e/--encapsulated` option to indicate that the generated MICD file must encapsulate the FDEF model and the initial input model.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0060

ADCN-FDEF accepts the `-n/--noswap` option to force the use of the option /NO_SWAP for GAC execution.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0070

ADCN-FDEF accepts the `-v/--version` option to specify the version of the MICD to produce. If the `--encapsulated` option is not used, this option must be used.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0080

ADCN-FDEF accepts the `-a/--adcn` option to indicate that ADCN-Simu must be executed.

If this option is used, the option --mnemo must also be used and an ADCN file of .ncd or .csv format must be given as a parameter of the command line.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0090

ADCN-FDEF accepts the `-g/--generate_dec` option to indicate that a dec file must be generated.This file is need to compile the model with MAMBO.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0100

ADCN-FDEF accepts the `--micd_m2633_2` option to indicate that the generated MICD must follow the AP2633 Issue C rules (M2633.2).

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0110

ADCN-FDEF accepts the `-f/--filt_network_B` option to indicate that the network B shall be ignored for AFDX format. Ports that are exclusively defined on this network will not be treated by the tool.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0120

ADCN-FDEF accepts the `-x/--xml_only` option to indicate that generation code shall not be done (i.e. GAC and ADCN-Simu shall not be called) and that the xml folder shall not be deleted at the end of the execution.

#EndText
#Verify Test

E_STL_FDEF_CMD_ADCN_FDEF_0130

ADCN-FDEF accepts the `-k/--keep_xml` option to indicate that deletion of xml files shall not be done at the end of the generation.

#EndText
#Verify Test

5.5.8 Generation of a .dec file and a def.h file

E_STL_FDEF_DECDEF_ADCN_FDEF_0010

If the –g option is used, ADCN-FDEF generates a model_ID.dec file which contains all consumed and producted ports.

#EndText
#Verify Test

E_STL_FDEF_DECDEF_ADCN_FDEF_0020

If the –g option is used, ADCN-FDEF generates a model_ID_def.h file which contains data types definitions.

#EndText
#Verify Test

6 Operational requirements

This paragraph describes the operational requirements expressed in the form of specification constraints.

These constraints will guide later choices, especially those made during design.

6.1 Hardware requirements

E_STL_FDEF_MAT_ICD2XML_0010

The xml file generator ICD2XML must be capable of operating on DEC (TRUE 64 OS), SUN, Windows 2000 PC and Linux PC platforms.

#EndText
#Verify Test

E_STL_FDEF_MAT_GAC_0010

The FDEF code generator must be capable of operating on DEC (TRUE 64 OS), SUN, Windows 2000 PC and Linux PC platforms.

#EndText
#Verify Test

E_STL_FDEF_MAT_GAC_0020

The code generated by FDEF must run on DEC (TRUE 64 OS), SUN, Windows 2000 PC and Linux PC platforms.

#EndText
#Verify Test

E_STL_FDEF_MAT_ADCN_SIMU_0010

The ADCN-Simu code generator must be capable of operating on Windows 2000 PC and Linux PC platforms.

#EndText
#Verify Test

E_STL_FDEF_MAT_ADCN_FDEF_0010

The ADCN-FDEF code generator must be capable of operating on Windows 2000 PC and Linux PC platforms.

#EndText
#Verify Test

6.2 Software requirements

6.2.1 Limitations

E_STL_FDEF_LIM_ICD2XML_0010

The FDEF input XML files are assumed to be valid with regard to the description of the AFDX payloads, ARINC 429 words and CAN messages. Only tests on the syntax and the coherence of the structure of the XML files shall be done by FDEF.

#EndText
#Verify Test

6.2.2 Volumetry requirements

None.

6.2.3 Ergonomic requirements

None.

6.2.4 Reuse of existing process requirements

E_STL_FDEF_EX_GAC_0010

For read and exploitation of the configuration tables in XML language, SAX API implementation of Xerces type shall be used.

#EndText
#Verify Test

E_STL_FDEF_EX_GAC_0020

The code generated by FDEF must be capable of operating within ASPIC and within DSS; it must therefore comply with the coding rules related to these workshops.

#EndText
#Verify Test

E_STL_FDEF_EX_ADCN_SIMU_0010

The code generated by ADCN-Simu must be capable of operating within ASPIC and within DSS; it must therefore comply with the coding rules related to these workshops.

#EndText
#Verify Test

E_STL_FDEF_EX_ADCN_FDEF_0010

The code generated by ADCN-FDEF must be capable of operating within ASPIC and within DSS; it must therefore comply with the coding rules related to these workshops.

#EndText
#Verify Test

6.3 Quality requirements

Not applicable.

6.3.1 Maintainability requirements

Not applicable.

6.3.2 Reliability requirement

Not applicable.

6.3.3 Robustness requirements

E_STL_FDEF_ROB_GAC_0010

The GAC tool must be based on tested components and APIs. This must especially be the case for the loading and the exploitation of the XML tables and the conversion functions.

#EndText
#Verify Test

E_STL_FDEF_ROB_GAC_0020

If an incoherence is detected in the structure of the XML input files, a message must be given in the FDEF.log log file.

#EndText
#Verify Test

6.3.4 Upgradability requirements

None.

6.3.5 Security and confidentially requirements

Not applicable.

6.3.6 Traceability requirement

Not applicable.

6.3.7 Precision requirement

Not applicable.

7 Installation and configuration requirements

E_STL_FDEF_INST_0010

For the A/C0 simulators (operating under ASPIC), the generated code shall be exploited with the ATPRO-CC workshop.

#EndText
#Verify Analysis

E_STL_FDEF_INST_0020

On PC platform, Microsoft visual C++ 6.0 shall be used to exploit the generated code.

#EndText
#Verify Analysis