

## ASIML-1553: A MARKUP LANGUAGE FOR MIL-STD-1553B AVIONICS BUS SCHEDULER AND INTERFACE CONTROL

Neeta Trivedi  
Scientist

Aeronautical Development Establishment  
Defence Research and Development Organisation  
New Thippasandra Post  
Bangalore-560 075, India  
Email : neeta@ade.drdo.in

### Abstract

*Modern avionics is highly software intensive with various software components residing on different subsystems interfaced through avionics bus. Avionics system designer is responsible for defining the bus frame timings, message transmission schedules and details of interfaces. These are translated into design and finally into source code by the designers of individual subsystems. Growing complexity of avionics software coupled with frequent changes has necessitated use of highly layered architecture with tight partitions and open standards in order to obtain better maintainability, testability, portability and ease of upgrade. Textual configuration files offer benefit over embedding the configuration details into the code in terms of development, testing and certification efforts. XML is emerging as the worldwide standard for defining configuration files. This paper proposes ASIML-1553, a markup language for defining configuration files for avionics bus scheduler and interface control for MIL-STD-1553B bus. Data from a typical combat aircraft are taken as case study to discuss the advantages ASIML-1553 offers in terms of turn-around time, quality and reliability. ASIML-1553 can easily be adapted for interfaces other than MIL-STD-1553B such as ARINC-429 or Ethernet.*

**Keywords:** Avionics, MIL-STD-1553B, ARINC, Ethernet, XML, Software Quality and Reliability, Open Standards

### Introduction

Avionics is taking up increasingly large share of the cost and schedule of modern aircraft programmes. From the modest 100-200 4-8 bit words of assembly code for fire control in the fighters of 1960s [1] to the billions of 32-bit words of higher level language code for flight control, electronic warfare, cockpit displays, engine management, stores management, mission management, built-in-test, in-flight planning and so on, software size and complexity is growing substantially. Declining hardware costs are not able to compensate for the escalating software costs [2].

Technological advancements and changing threat perceptions cause the avionics segment to evolve continuously. Newer and better equipment gets added; older ones

removed. New functionality is implemented, functional integration happens, sensors get added that provide newer inputs, avionics gets upgraded.

The flexibility software offers for accommodating changes is a significant attraction to have as much of functionality built in to software as possible. The growing complexity of avionics software coupled with frequent changes calls for robust architecture, tight partitioning and open interfaces to minimize perturbations to the system and also reduce testing efforts.

Digital serial multiplexed communication buses are being widely used for interfacing aircraft subsystems due to the advantages they offer in terms of reduced interconnects, driver-receiver electronics and weight. The bus

most commonly used in most military aircraft, missiles, and many other aerospace platforms is the United States time-division-multiplexed digital serial MIL-STD-1553B [3]. The standard specifies the physical layer and medium access sublayer specifications for the interface. The MIL-STD-1553B standard follows command-response protocol with one node identified as the Bus Controller (BC) and others as Remote Terminals (RT). BC controls transfer of all data on the bus.

For a given aircraft programme, avionics systems designer specifies the bus frame timings, message scheduling scheme and the BC-to-RT and RT-to-RT interface control details. These details are communicated to the designers of individual subsystems where they get translated into design and finally into source code. The frame timings and the message scheduling schemes (frequency, frame assignment) form part of what is typically known as 'Bus Scheduler' or BCX for short. There may be various phases of aircraft operation such as Initialization, Operational Flight Programme, Post-flight Data Retrieval etc. Some messages may be common across these phases, some may be unique to a particular phase.

The details of the messages in terms of the sender ID, receiver ID, transmit subaddress, receive subaddress, number of words as well as contents of each word and their meanings form part of what is typically called the Interface Control Details (ICD).

Any addition, removal or upgrade of subsystems or their functionalities result in changes to the BCX and/or the ICD, motivating the need to absorb these changes with minimal impact. An effective way of absorbing these changes in least perturbing manner is to use configuration files for structured static data that are accessed by device managers. Configuration files provide a concise method for defining configuration in implementation-independent manner. These are not to be accessed by the application program; however, they are not built as part of the device drivers, either. EXtended Markup Language (XML) [4] is emerging as the language of choice for defining configuration files.

This paper proposes a COTS way of implementing the device-independent 1553B I/O software component in avionics subsystems viz. using XML to define configuration files. The author proposes ASIML-1553, a Markup Language for BCX and ICD for MIL-STD-1553B bus. Any XML parser can be used in the preprocessing step for

parsing ASIML-1553 file and generating an internal data structure to be used by the application. ASIML-1553 also provides constructs for specifying typical configuration details for MIL-STD-1553B interface devices. This helps in initialization of the device.

Some of the lifecycle details related to the BCX and ICD of a typical combat aircraft are taken as case study to demonstrate the power of the language and the interfacing logic.

With the huge installation base as well as the robustness characteristics, MIL-STD-1553B plays a vital role in a wide variety of military systems. The interface of choice in case of civil aircraft industry has been ARINC-429. While MIL-STD-1553B and ARINC-429 are still considered good for control applications, high-speed networking solutions such as Ethernet are also fast emerging as alternatives for bandwidth-critical applications. ASIML-1553 can be adapted to such alternatives with ease.

Section 2 describes related work and standards. Section 3 shows functional layers of typical avionics software and discusses how changes are being incorporated in most avionics systems at present. Section 4 describes the proposed approach, details the features of the proposed language and discusses the change cycle in the new scenario. A theoretical quantitative analysis of benefits is presented in section 5. Conclusions are drawn in section 6.

### Related Work and Standards

The eXtensible Markup Language (XML) [4] is a subset of SGML (ISO 8879). The XML standard is maintained by the World Wide Web Consortium (W3C). In this paper, XML (V 1.0) has been used to describe the configuration data. XML-Schema (V 1.0) is used to define the format of the XML data.

Much has been discussed in the literature about use of XML for defining the configuration details. ARINC Specification 653 uses XML to specify the scheme for defining ARINC 653 configuration data [5]. For MIL-STD-1553B, software for bus protocol validation and bus traffic analysis systems [6,7] use XML for defining configuration files. While these can be viewed as MAC and network layer details, ASIML-1553 can be roughly viewed as the language for the transport layer, where header, data words, sequencing etc. is defined. The XML can be used during developmental testing by Bus Monitors to display the data along with its semantics.

To the best of the knowledge of the author, no schema has been specified for defining MIL-STD-1553B interface at this layer.

**Bus Interface Design**

The avionics bus interface design involves designing of the bus scheduler and the interface details by the avionics system designer and the device configuration and I/O design by the designer of individual units (typically called Line Replaceable Units or LRU).

**Bus Scheduler**

As discussed earlier, the bus scheduler must specify the frame timings and message scheduling scheme. Although the naming conventions may vary, most avionics programmes define a ‘Minor Frame’, which is the smallest time interval or the highest frequency of data. Once the frequency of all the data sets is established, the minor frame duration can be taken as the greatest common divisor of all the frequencies.

Another term often used is ‘Major Frame’. A number of minor frames constitute a major frame. This number is usually taken such that every message occurs at least once in a major frame. Once the frequency of all the data sets is established, the major frame duration can be taken as the least common multiplier of all the frequencies.

The next part to be included in the BCX is the scheduling of various messages in each frame for each of the avionics operational phases. Usually this is done in such a manner as to group related elements together and yet achieve a balanced load in different minor cycles. An example instance of BCX is shown in Table-1. Note that an aircraft may have more than one avionics buses, for increased reliability, isolation of unrelated messages, avoiding bus overloading, or because one bus may have finished the maximum possible 30 ‘remote terminals’. The

<b>Table-1 : An Example of a BCX</b>			
Bus 1, INIT Frame			
Minor Frame 0	Minor Frame 1	Minor Frame 2	Minor Frame 3
BC-LRU1 LRU2- LRU3 LRU3-BC	LRU5-BC	BC-LRU1 LRU4- LRU3	BC-LRU4
(Assuming 4 minor frames in a major frame)			

BCX provides details of scheduling on each of these buses for each operational phase.

**Interface Control Details**

The ICD is responsible for providing details on the interpretation of each message.

MIL-STD-1553B defines 3 types of messages: Transmit, Receive, and Mode Codes. Transmit/receive command word contains RT address, subaddress and number of words to transmit/receive. Mode Code command word contains RT address, transmit/receive flag, mode code type and mode code number. Maximum number of words for any command are 32; however, receive commands can be indexed, in which case multiple sets of words are received on the same subaddress and are multiplexed based on some ID.

The ICD lists out the following for each transmit/receive message.

- Source (Bus Controller or an RT address)
- Destination (Bus Controller, an RT address or broadcast)
- Transmit and receive subaddresses (1553B specifies 30 subaddresses plus two reserved for indicating ‘Mode Code’, discussed later)
- Mux ID (if the message is multiplexed i.e. multiple messages arriving on a single subaddress. This ID is part of the message and is handled by the application. The 1553B device does not recognize this as a control)
- Number of words in the message
- For each word, the details of its contents including its type (e.g. unsigned number, 2s complement signed number, binary-coded decimal, discrete, composite), resolution value where applicable and so on. An example is given in Table-2 and 3.

**Device Configuration and I/O**

The descriptor space in the 1553B devices must be configured to handle the different messages, the required number of words and required number of indexing for various commands. This forms part of the device-dependent I/O. The device-independent I/O layer must perform sanity checks on the data in terms of the messages arrived in a particular cycle, number of words in the messages,

Message ID	LRU3-BC		
Source	LRU3		
Dest	BC		
Word count	4		
Rate	12.5 Hz		
Description	Message from LRU3 to Bus Controller, Gives health status		
Message Details	Word No.	Word Name	Description
	00	Link 1	Link 1 Status
	01	Port 1	Port 1 Status
	02	Memory	Memory Status
	03	Temp	Device Temperature

correctness of message status words and so on. This requires knowledge of the BCX and ICD at this layer.

### Functional Layers in Avionics Software

Avionics software is typically designed in a layered form as given in Fig.1. The 1553B protocol part is a component in the device-dependent I/O layer and does not call for changes once the bus I/O devices are fixed.

Typical interaction between the I/O modules and application layer modules (possibly through software service layers) is as shown in Fig.2, where the interaction between application layer and device dependent I/O is quite infrequent. Individual modules have their own data structures to store relevant local details.

The aircraft-specific scheduling and interface details change more frequently. These are out of scope of the device-dependent layer and hence can be easily decoupled. Of course, these changes are always application-specific and hence require changes to application layer modules. However, in the present scheme, these changes (for example change in number of words for a receive message, multiplexing on a given subaddress etc.) additionally require changes in the device-independent I/O modules. Assuming SDLC standard to be RTCA-DO-

Word ID	LRU3-BC/00		
Word Name	LRU3-Link 1		
Signal Type	Binary Code Decimal		
Units	NA		
Resolution	NA		
Remarks	-		
Word Details	Bit No	Variable Name	Description
	00-07	Link1-Local Device	Failure Code
	08-15	Link1-External	Failure Code
Word ID	LRU3-BC/03		
Word Name	LRU3-Temp		
Signal Type	2s Complement		
Units	Degrees Fahrenheit		
Resolution	0.1		
Remarks	-		
Word Details	Bit No	Variable Name	Description
	00-15	Decive_Temp	Temperature

178B [8] level C or above, these changes translate into significant efforts towards documentation, implementation, testing, change cycle management and verification.

Both BCX and ICD are highly structured data items and hence make perfect candidates for definition through XML schemas. The pattern inherent in the BCX and ICD can be exploited for reduction in time and efforts, and that is the theme of this paper.

### ASIML-1553

The ASIML-1553-Schema defines the structure of the data needed to specify MIL-STD-1553B interface configuration. The schema is the reference standard to which instance files are defined. It consists of tagged pairs that describe the attributes and their relationship to the whole.

The graphical view of ASIML-1553 schema is presented in Appendix-A. The view was generated using the Altova's XMLSpy [9] version 2010 release 3 sp1. Appendix-B specifies the structure of the ASIML-1553 instance file in which the configuration data parameters are specified.

### ASIML-1553 Extensions

Like most other XML schemas, the ASIML-1553 schema is also extensible; the designers can extend it for a particular implementation. For example, a designer using a UTM 1553B device can select 'ping pong' enable or disable for individual subaddresses. All the device-specific details can appear in the extensions to ASIML-1553.

### ASIML-1553 Benefits

Using ASIML-1553 for interface specifications allows I/O modules to be completely decoupled from the changes in application software. Given that most aircraft programmes follow rigid norms for lifecycle process, changes result in a lot of documentation, testing and V and V efforts. Most importantly, by reducing requirement for source code changes, one is almost always talking high quality and by reducing source code changes, one is almost always talking high reliability.

Everything comes at a price. Generality always comes at the cost of execution speed. However, two aspects speak in favor of ASIML-1553. First, most of the configuration settings are processed during the initialization phase and hence do not affect the real-time performance. Second, the effect during real-time operations is a small price to be paid given today's processing speeds and capabilities, considering the enormous saving in efforts and confidence in reliability.

### Effectiveness Analysis

Changes to the bus interface are very common in developmental aircraft programmes as well as avionics upgrades. The frequency and volume of changes may get reduced as the programme matures, but it is at this stage that even small changes can prove costly and hence the weighted impact of changes can be arguably the same.

We take as case study a developmental programme with which the author was associated. Cockpit displays are the front-end and the only source of information for the pilot. In a glass-cockpit system typical of modern aircraft where displays are managed through software, changes to

any avionics system invariably call for changes in display software requirements. In a span of 5 years, six changes were made to the ICD. Table-4 provides some details of these changes.

Table-5 provides an estimated number of lines of code to be modified, added or deleted because of the changes listed in Table-4.

The effort estimation includes changes to relevant documents, source code, unit testing, integration testing, verification and validation and QA activities. By using ASIML-1553 for configuration and using the parsed tree appropriately, the effort can be reduced to about 10% of the original, which will essentially involve modification

Sr. No.	Addition	Modification	Deletion
1	46	65	73
2	4	-	-
3	-	15	-
4	-	17	-
5	30	13	12
6	8	10	-

Sr. No.	No. of changes to device-independent I/O (blocks of source code, average 5 lines per block)	Estimated Effort for DO-178B Level B Process (including documentation, testing, V & V, QA)
1A	16	4 man - months
1M	10	2 man - months
1D	30	5 man - months
2A	4	1 man - month
3M	0	0
4M	0	0
5A	10	3 man - months
5M	5	1 man - month
5D	10	3 man - months
6A	10	4 man - months

A : Addition; D : Deletion; M : Modification;  
Sr.No. from Table-4

to the configuration file, verification of the file, configuration control, and integration testing.

The saving in efforts is also due to the fact that the traditional programs are in a computer language, mostly C/C++, which require certain skill level to handle. For the XML, editors are available e.g. [9] that allow easy entry of data with no special skills involved. The XML editors also perform syntax checks thereby ensuring data integrity. The same holds good of verification; verification of software design and code requires special skillset whereas data files can be verified by less skilled personnel.

Note that the efforts outlined here only mention about those for the device-independent I/O modules (which the ASIML is trying to replace) and not the overall changes to the application software. The efforts involved at application layer stay the same as earlier.

### Responsibilities

This section describes an example how the ASIML-1553-Schema can be developed and used for defining the RT interface configuration files. There are three major steps involved in defining and using the schema.

- The avionics systems group defines the ASIML-1553-Schema.
- The avionics system group will also create and validate the ASIML-1553 instance file for the various remote terminals. The instance is based on the ASIML-1553-Schema. A validation tool may be used to verify that the instance file is well formed and valid against the ASIML-1553-Schema.
- The RT software implementer may develop a parser or use one of the standard parsers. The parser will use the ASIML-1553-Schema as the reference for converting an instance file into a format used by the application implementation.

### Adapting to Alternate Interfaces

Usage of ARINC or Ethernet for defining the BCX and ICD will call for changes more in the BCX. The controls used by control words may have different syntax. For example, the header used for a packet on Ethernet may write the source and destination addresses in a different format at a different location in the header. The schema does not change substantially in this case; the attributes of

the variables probably will. The parser also remains the same. The program that reads the parsed data to generate data for internal use needs to be written accordingly.

### Conclusions

An XML-based method of defining the device-independent I/O part of avionics software is described. The schema is extensible; device-dependent configuration details can be added as extension to the schema. The schema can be easily adapted to other interfacing standards such as ARINC or Ethernet. Based on the author's experience in a developmental aircraft programme, a comparison of effort levels with and without the XML-based approach is drawn. The new approach does not only reduce effort levels but also provides higher level of confidence in the process, since most part of the software is not altered manually.

### Acknowledgement

The author has worked on MIL-STD-153B at the Cockpit Display Group of ADE, and she wishes to thank the members of the group for their cooperation during the work. The valuable suggestions on this paper from Mr. Suresh Kumar, Ex-Project Director at ADE are hereby duly acknowledged. The author is also thankful to Director, ADE for his permission to publish the paper.

### References




1. Christine Anderson and Merlin Dorfman (Editors), "Aerospace Software Engineering, Progress in Astronautics and Aeronautics", (Editor-in-chief A Richard Seebass), Vol.136, 1991, Publisher American Institute of Aeronautics and Astronautics, ISBN 1-56347-005-5, 629.7:681.31.06.
2. Neeta Trivedi and Suresh Kumar., "Development of Large Real-time Avionics Software: Our Experiences", Proceedings of 3<sup>rd</sup> APCATS, China.
3. MIL-STD-1553B (Notice 2): "Digital Time Division Command/ Response Multiplex Data Bus", 08 September 1986.
4. <http://www.w3.org/XML/>, last accessed 19 June 2011.
5. <http://www.linuxworks.com/solutions/milaero/arin-653.php>, last accessed 19 June 2011.

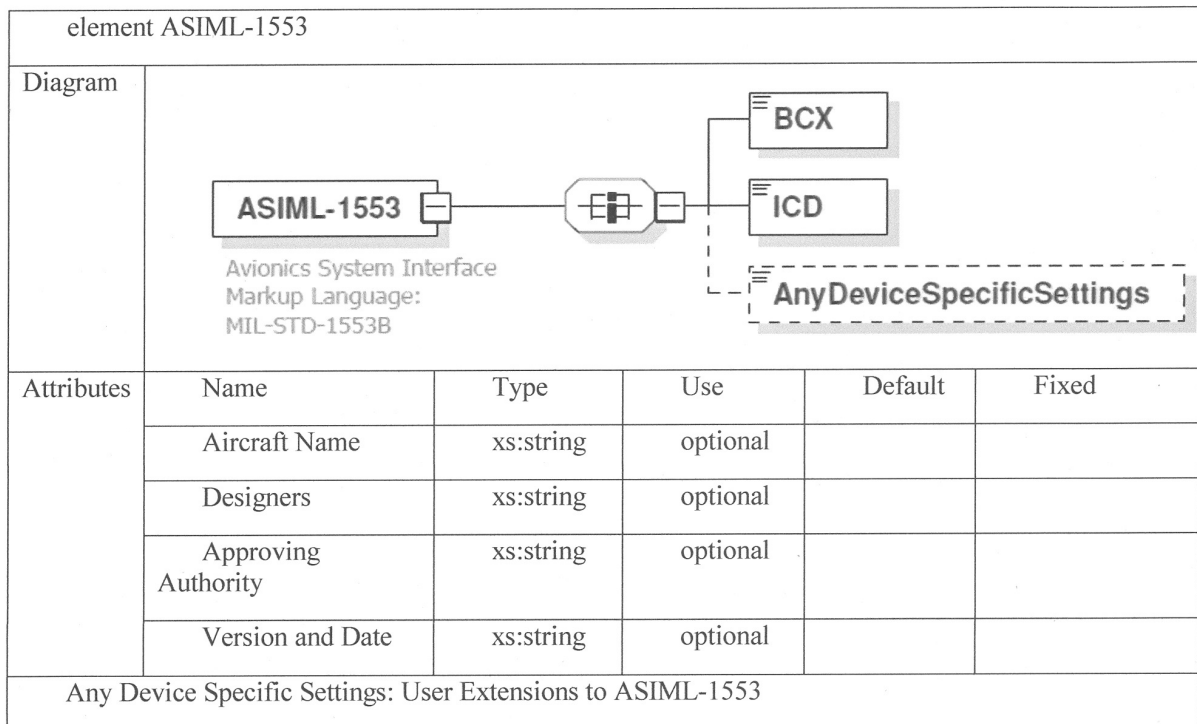
- 6. [http://www.sitaltech.com/1553\\_test.asp](http://www.sitaltech.com/1553_test.asp), last accessed 19 June 2011.
- 7. <http://www.altadt.com>, last accessed 19 June 2011.
- 8. <http://www.do178site.com/>, last accessed 19 June 2011.
- 9. <http://www.altova.com/xml-editor/>, last accessed 19 June 2011.

**APPENDIX-A**

**ASIML-1553 Schema: Graphical View**

The following symbols have been used in the figures in this appendix. The symbols are artifacts of the tool XMLSpy and not part of the XML standard.

Symbol	Meaning
	Corresponds to the XML definition xs:sequence element. All child elements must appear in sequential order.
	Corresponds to the XML definition xs:choice element. Only one of the child elements can be chosen.
	Corresponds to the XML definition xs:all element. All the child elements can appear in any order.



element ASIML-1553/BCX					
Diagram					
Attributes	Name	Type	Use	Default	Fixed
	Bus ID	xs:string	required		
	Designers	xs:string	optional		
	Approving Authority	xs:string	optional		
	Version and Date	xs:string	optional		
	Revision History	xs:string	optional		

element ASIML-1553/BCX/MessageComposition					
Diagram					
Attributes	Name	Type	Use	Default	Fixed
	Remarks	xs:string	optional		



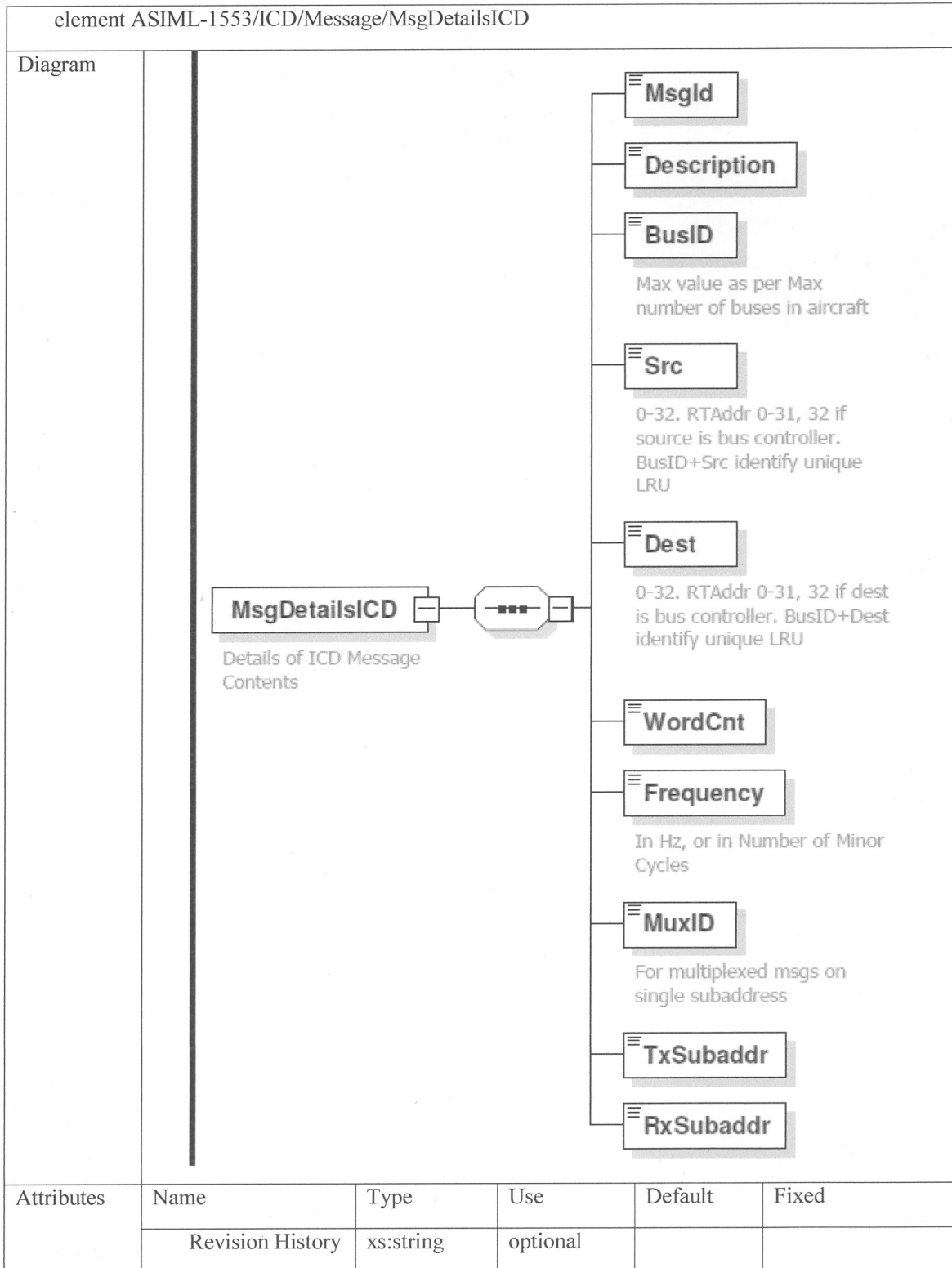
element ASIML-1553/BCX/MessageComposition/MajorFrame					
Diagram	<p>All messages in one major frame</p>				
Attributes	Name	Type	Use	Default	Fixed
	Remarks	xs:string	optional		

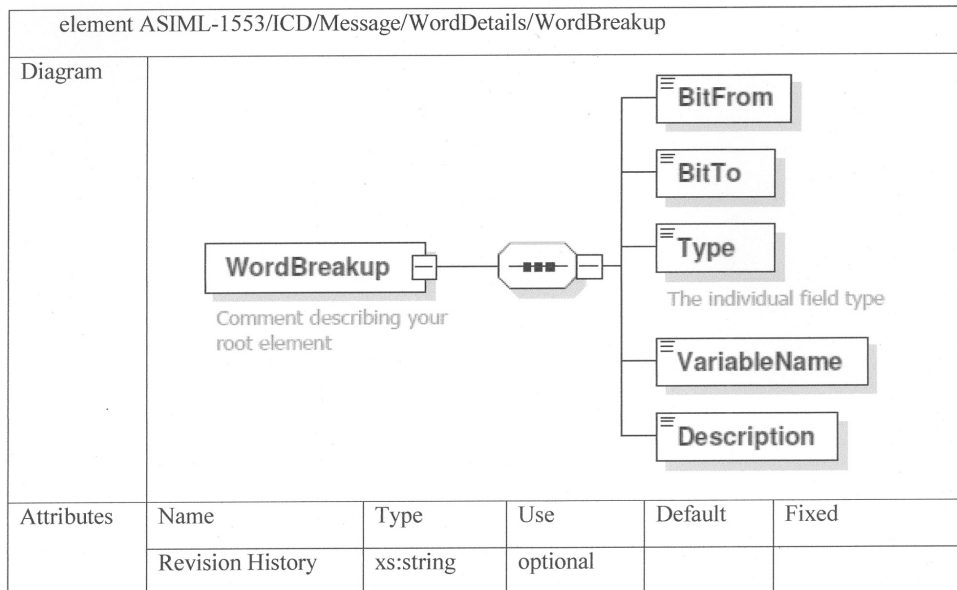
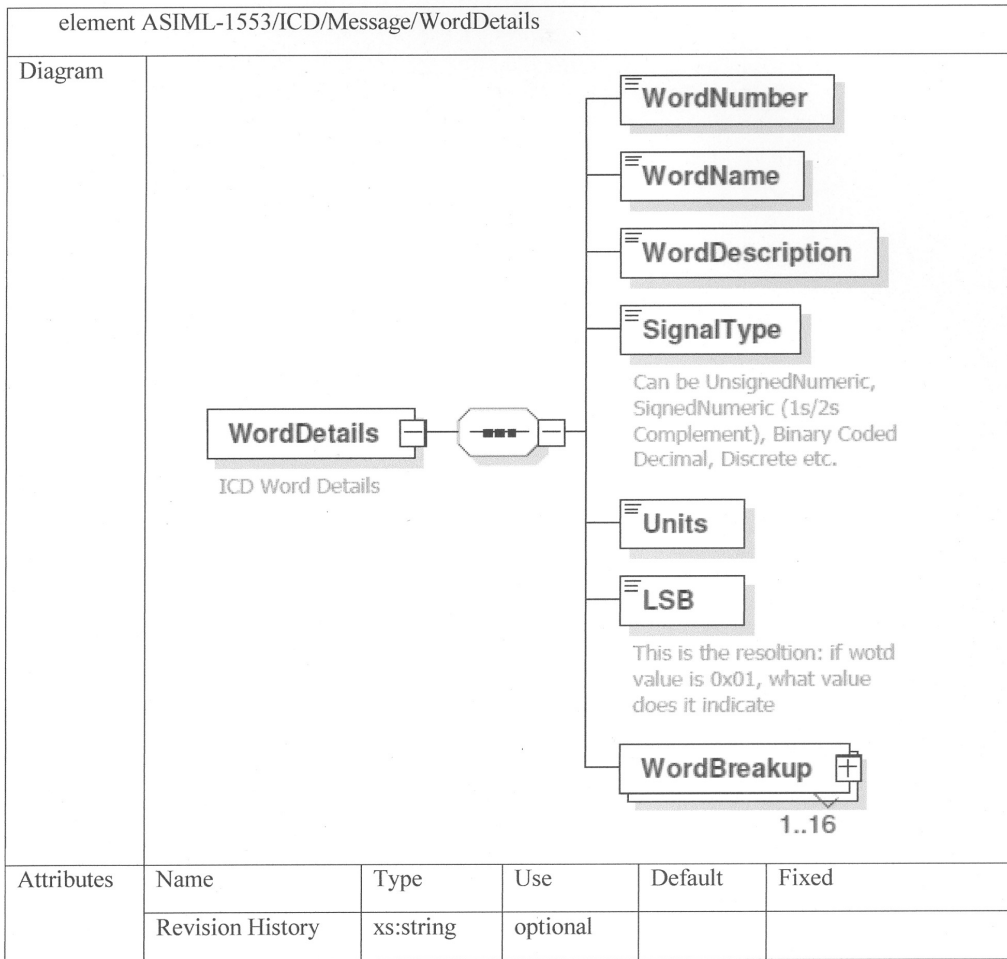
element ASIML-1553/BCX/MessageComposition/MajorFrame/MinorFrame					
Diagram	<p>Comment describing your root element</p>				
Attributes	Name	Type	Use	Default	Fixed
	Remarks	xs:string	optional		

element ASIML-1553/BCX/MessageComposition/MajorFrame/MinorFrame/MsgDetail					
Diagram	<p>Details of the 1553B Message. Msg Type is implicitly known from TxOrRx and Subaddr fields</p>				
Attributes	Name	Type	Use	Default	Fixed
	Remarks	xs:string	optional		

element ASIML-1553/ICD					
Diagram					
Attributes	Name	Type	Use	Default	Fixed
	Designers	xs:string	optional		
	Approving Authority	xs:string	optional		
	Version and Date	xs:string	optional		
	Revision History	xs:string	optional		

element ASIML-1553/ICD/Message					
Diagram					
Attributes	Name	Type	Use	Default	Fixed
	Revision History	xs:string	optional		









```

<xs:element name="WordCnt"/>
<xs:element name="Frequency">
  <xs:annotation>
    <xs:documentation>In Hz, or Minor Cycles</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="MuxID">
  <xs:annotation>
    <xs:documentation>For multiplexed msgs on single
      subaddress</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="TxSubaddr"/>
<xs:element name="RxSubaddr"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="WordDetails" maxOccurs="32">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="WordNumber"/>
      <xs:element name="WordName"/>
      <xs:element name="WordDescription"/>
      <xs:element name="SignalType">
        <xs:annotation>
          <xs:documentation>Can be UnsignedNumeric, SignedNumeric
            (1s/2s Complement), Binary Coded Decimal, Discrete
            etc.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Units"/>
      <xs:element name="LSB">
        <xs:annotation>
          <xs:documentation>This is the resolution: if wotd value is 0x01, what
            value does it indicate</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="WordBreakup" maxOccurs="16">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BitFrom"/>
        <xs:element name="BitTo"/>
        <xs:element name="Type">
          <xs:annotation>
            <xs:documentation>TypeOfVariable e.g. Discrete/BCD
              etc.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="VariableName"/>
        <xs:element name="Description"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

```

