



To SDL Subcommittee **Date** December 29, 2009

From Scott L. Smith **Reference** 09-164/SDL-55 cjm
slsmith@arinc.com
tel +1 410-266-2805

Subject **Strawman Circulation**
Strawman for ARINC Project Paper 838: Loadable Software Parts Using XML

Summary The attached strawman is preliminary text submitted by members of the Software Data Loader (SDL) Subcommittee that provides guidance to using extensible markup language (XML) in software data loading. The intent is to provide methods and processes to allow for human-readable header file elements to benefit the airline maintenance users.

The SDL Subcommittee will review this strawman and any comments received at their next meeting. The next meeting is tentatively scheduled for the first week in March 2010 in the Los Angeles, California, area. The meeting details will be published as soon as they are confirmed.

Action Please review the strawman and notify Scott Smith in writing (e-mail) if you have any comments.

cc FLS Working Group

STRAWMAN
OF
ARINC PROJECT PAPER 838
LOADABLE SOFTWARE PARTS USING XML

This strawman dated: December 29, 2009

This document is based on material submitted by various participants during the draft process. Neither AEEC or ARINC has made any determination whether these materials could be subject to the claims of patent or other proprietary right by third parties, and no representation or warranty, expressed or implied, is made in this regard. Any use or reliance on this document shall constitute an acceptance hereof "as is" and is subject to this disclaimer.

This is a working paper for the AEEC. It does not constitute air transport industry or ARINC approved policy, nor is it endorsed by the U.S. Federal Government, any of its agencies or others who may have participated in its preparation.

ARINC SPECIFICATION 838
TABLE OF CONTENTS

1.0	INTRODUCTION	3
1.1	Purpose of Document.....	3
1.2	Overview.....	3
1.3	Objective.....	3
1.4	Scope	4
1.5	Related Standards.....	5
1.5.1	Non-ARINC Standards	5
1.6	Support Tools	6
2.0	OVERVIEW	7
2.1	Origins of ARINC 838 Format (Remove from final draft before publishing)	7
2.2	Foundations.....	8
2.3	Tradeoff	8
2.4	Benefits.....	8
2.5	Target Use Cases.....	8
2.5.1	Small Target	8
2.5.2	High Design Assurance Level Target	9
2.5.3	Low Design Assurance Level Target	9
2.6	Implementation	9
2.6.1	Transition of ARINC 665 Legacy Parts and Media	9
2.6.2	ARINC 615A Data Loader Implementation.....	9
2.6.3	ARINC 615A Target Loader Implementation	9
ATTACHMENT 1	GENERIC DEFINITION OF PART DEFINITION FORMAT	11
ATTACHMENT 2	XML FORMAT.....	17
ATTACHMENT 3	BINARY FORMAT	22
APPENDIX A	LIST OF ACRONYMS	33
APPENDIX B	GLOSSARY.....	34
APPENDIX C	XML CONVENTIONS	35
APPENDIX D	XML SCHEMA	36
APPENDIX E	XML EXAMPLE – SMALL FILE.....	39
APPENDIX F	XML EXAMPLE – COMPLETE FILE.....	40
APPENDIX G	BINARY EXAMPLE – SMALL FILE	42
APPENDIX H	BINARY EXAMPLE – COMPLETE FILE	43
APPENDIX I	COMPARISON TO OTHER SOFTWARE STANDARDS	44

CHANGE LOG

CHANGE LOG

When	Who	What	Why
2009-11-02-IV	Carsten Schweigert	<ul style="list-style-type: none"> - Added check values for files. Made optional. - Made load part flags optional - Load type ID: made it a string. Airbus would like this to be a string (FIN), Boeing uses numbers. - File-loadable: needed clarification - File-location: made optional. Optional if no subdirectory is used, and all files are in part root directory - File-description: made optional - File part number: made optional if not used. - File length: renamed to "file-size" - Put in an XML example for a minimal part (only mandatory fields) - Put in an XML example for a full part (including all optional fields) - Added section for binary example for full and minimal part. - CRCs: added requested checksums SHA-1, CRC32, CRC64, and allow for no checksum. - Clarified confusion between "uploadable" and "file-loadable" fields. - Indicated that partflags and file-loadable are scrutinized for removal. - Corrected errors in schema, checked schema - Corrected match between schema and binary notation - Added description for loader: how LUR/LUH/LUX are used. - Added description that fields of binary file to compute checksum are used. - Added description that the bin file does not cover xml file. XML load description file has same content as binary file. Lists bin file, and has same CRCs as binary file. - Added location and file naming of LUH and LDF file - Document limitations <ul style="list-style-type: none"> o Lacking some nice diagrams o Rules for whitespace/ordered list not defined o Bin example file to be added (with all fields, and without optional fields) o Rules for integer number representation to be added. 	Review by SDL 2009-04, Portland, U.S.A.

CHANGE LOG

		<ul style="list-style-type: none">○ Binary example needs update○ Binary template needs update	
--	--	--	--

1.0 INTRODUCTION

1.1 Purpose of Document

The significant increase in field loadable software parts in today's airplanes (up to 1000 software parts to be managed) results in increasing levels of automated electronic processing. Managing electronic software parts as opposed to manual processing of software parts distributed on physical transport media includes automatic electronic comparison, configuration management, packaging and un-packaging.

The Electronic Distribution of Software (EDS) group updated the EDS ARINC Standard from complex binary data structures to XML (crates).

This report intends to perform the same transition for Software Data Loader (SDL) Standard for Loadable Software Parts (LSPs).

The end result is complete harmonization of data formats for crates as well as for electronic distribution and packages within crates, allowing for automatic processing, and inspection, using standardized tools.

1.2 Overview

This report describes the principles and rules for packaging of data files into LSPs.

This SDL Standard is intended to promote simple, consistent, efficient, and automated packaging of data files as loadable software parts using proven, advanced methods which integrate seamlessly with Field Loadable Software (FLS) and EDS standards.

This is a modern SDL package definition standard developed by the Software Data Loader Subcommittee of the AEEC. ARINC Report 665 defined the standards for electronically packaging aircraft loadable software files into software parts (loadable software parts), creating software load batch files, and bundling of software parts on physical media sets (media set parts).

Significant advancements have been made since the original standard was released:

- Global standardization of XML format
- Selection of XML by EDS group to define crates for electronic distribution
- Industry-wide trend to migrate towards electronic distribution

This report focuses on the migration of complex, binary ARINC 665 header files to simple XML software part definition. Future refinements to this report are expected to strategically incorporate emerging technologies, methodologies, and designs as necessary to improve LSP packaging processes.

1.3 Objective

The main objective of this report is to:

- Standardize electronically distributable ARINC 665 loadable software parts, to support both packaging for electronic distribution by means of ARINC 827 electronic crates, and ARINC 665 transport media

1.0 INTRODUCTION

- To allow for humanly-readable files for electronic transmission and electronic validation (XML in ARINC Specification 838 format)
- To allow for completely XML-based (compatible with ARINC Specification 665) load packaging of software for higher-level applications
- To provide a solution that is suitable for loaders, electronic distribution, and targets (qualification and certification)

The primary objective of this report targets electronic packaging of software (LSPs), using XML, and to provide formal definitions for those LSPs. A potential future objective of this report is the extension to also cover XML definitions for batch and media set definition files. The fundamental structure and definition of an LSP XML part is designed to accommodate:

- LSP distribution using EDS crates (standardized processing)
- LSP installation on target computers (certification aspects, simplicity)
- LSP installation by loaders (re-use of existing designs).

Manufacturers for certifiable software parts preferred binary definitions of file formats, in terms of field name, offset, length, and pointer, as reflected in the ARINC 665 standard. End users of software parts prefer standardized text file definitions for file formats. The objective of this report is to propose a definition for a simple software part which:

- Is a valid binary part, which can be processed by simple target loaders.
- Is a valid XML part, which can be read by standard parsers.
- Allows simple translation from binary format to XML and back

This report does not cover the same migration to XML for ARINC 665 media files. The extension for migration of ARINC 665 media files to XML in a revised version would depend on the assessed need and value for the industry.

1.4 Scope

The availability of standard XML software development tools will simplify the task of creating software parts for various customers.

- Support of loader needs and electronic distribution
 - Definition of simple, standardized ARINC software part description
 - Usage of XML-compliant subset to ease readability and electronic processing by COTS tools and methods
- Support of target needs
 - Definition of certifiable sub-set of XML notation that can be expressed by simple grammar
 - Definition of subset that supports processing by finite state machine, simple top-down or sequential parser
 - Definition of binary equivalent that can be translated into XML notation into binary notation using simple, cost-effective qualified tool.

Analogous to the practices of assembly of physical subcomponents into a hardware part, this standard applies the concept of a “part” to offer simple assembly, inspection, integrity check, and comparison for software files and related digital data.

1.0 INTRODUCTION

This report addresses

- Structural definition of an XML software part
- Integrity check for an XML software part
- Transformation from XML software part definition to binary content, and reverse.

1.5 Related Standards

Software parts are applied to aircraft with respect to a set of related standards and specifications. Related ARINC Standards include:

ARINC Report 615: *Airborne Computer High Speed Data Loader*

ARINC Report 615A: *Software Data Loader Using Ethernet Interface*

ARINC Report 664: *Aircraft Data Network*

ARINC Report 665: *Loadable Software Standards*

ARINC Report 666: *Electronic Distribution of Software*

ARINC Report 827: *Electronic Distribution of Software by Crate (EDS Crate)*

ARINC Report 667: *Guidance for the Management of Field Loadable Software*

ARINC Characteristic 763: *Network Server System (NSS)*

1.5.1 Non-ARINC Standards

ARINC Standards may refer to other industry standards and specifications. In all cases, it is intended that the most recent version of these publications are referenced. Examples of these include:

ATA Spec 2000: *e-Business Specification for Materials Management (Air Transport Association)*

ATA iSpec 2200: *Digital Certificate Policies (Air Transport Association)*

Electronic Security Infrastructure: *World Wide Web Consortium (W3C), XML Signature Syntax and Processing Recommendation*

Electronic Security Infrastructure: *World Wide Web Consortium (W3C),*

Encryption Recommendations issued from IETF/W3C XML

1.0 INTRODUCTION

1.6 Support Tools

This report does not endorse specific tools and processes to be used for XML software parts. Commercial entities are encouraged to apply this ARINC Standard in the manner they find optimal to develop and market tools and processes to meet and support objectives of this standard.

This report provides the foundation to create simple, cost-effective qualified tools to create binary and XML Load Definition Files.

2.0 OVERVIEW

2.0 OVERVIEW

This report describes the definition of a file format for field loadable software parts, containing information about data files and data file attributes (checksum, size, names), part attributes (part number, description, checksum). The ARINC Specification 838 format is a notation which can be translated into a binary file similar in structure to the binary file of ARINC Specification 665-3 and which can be translated into an XML file with identical fields and information.

Note: add diagram.

The file format subset can be read by COTS XML parsers and validation software tools without modification. The file format contains specific, additional restrictions to enable direct conversion between binary and XML load definition files.

A one-to-one relationship exists between the binary version and XML version of the XML part Load Definition File (LDF).

A load shall contain both the binary and XML formatted Load Part Definition file. The Load Part Definition files shall reside in the part root directory of the software part.

COMMENTARY

Although the binary and XML LDFs should be identical, in the case of software disagreement, the binary version should be considered the prime use case.

The one-to-one relationship between XML formatted Load Part Definition file and binary Load Part Definition file allows

- Target manufacturers to select if they implement a binary or XML-based processing of the load.
- To build an extremely simple, low-cost qualified tools to generate XML and binary Load Part Definition files.
- Airlines, Airframers, and third party application providers to use COTS libraries and COTS tools to process XML software parts.

2.1 Origins of ARINC 838 Format **(Remove from final draft before publishing)**

This report was generated by:

- Review of the fields in ARINC Report 665 and identification which fields should be kept
- Definition of a high-level grammar for the generic structure of the file.
- Mapping of the format-independent grammar to two different types of notations:
 - Binary notation, similar to format and concepts in ARINC 665-2/3.

2.0 OVERVIEW

- XML notation with XML schema, similar in format and concepts of ARINC 827.

COMMENTARY

The emerging use of XML in software should be considered using a new name or acronym, such as “LSP-X” to distinguish it from binary ARINC 665 formatted parts.

2.2 Foundations

The following assumptions form the basis for this report:

- A simple, qualified tool exists to create XML-based and binary Load Definition Files. Multiple vendors offer this tool.
- The load contains two Load Description files, one in binary format, one in XML format. The qualified translation tool generates cost-effective evidence of equivalence.
- There is no ambiguity in the description of the Load Definition file, based on usage of formal methods.
- The file equivalence allows using COTS XML tools and libraries to read the files.
- The availability of binary format allows low-cost transition of existing code to this standard.
- The availability of binary formats allows Design Assurance Level (DAL) systems to avoid the need to create a parser.

2.3 Tradeoff

Disadvantages of this definition compared to previous, purely binary definitions

- More memory space required in load (2 files, instead of 1).
- Qualified tool needed to perform automatic conversion.

2.4 Benefits

- Enables electronic processing using standard XML software packages by airframers, airlines.
- Enables electronic processing for creation and inspection of XML parts for XML crates.
- Loader: Not affected by change, loader does not need to parse header file.
- Target: Can choose which format is the most cost-effective for the LRU type.

2.5 Target Use Cases

2.5.1 Small Target

A severely memory and CPU constrained target will likely select the binary file. The target will perform the integrity check of the load, and then use the pointer architecture to navigate inside the file without the need to completely parse the

2.0 OVERVIEW

content. No high-level libraries are required, except file access and integrity check libraries.

2.5.2 High Design Assurance Level Target

A high design assurance level target which is not severely memory or CPU constrained can either choose certified XML libraries to validate and parse the XML Load Definition File, or select the binary Load Definition File for processing. Either approach does not require the manufacturer to write large amounts of certifiable code.

2.5.3 Low Design Assurance Level Target

A low design assurance level target which is not severely memory or CPU constrained can choose between binary and XML-based processing. Targets not affected by certification can use COTS XML libraries without the need for certification.

2.6 Implementation

2.6.1 Transition of ARINC 665 Legacy Parts and Media

Note: add diagram.

- XML part can reside in repositories and 665 media next to 665 parts as defined with current structure.
- Each XML and non-XML part should be put in its own directory, together with associated data files.

2.6.2 ARINC 615A Data Loader Implementation

Note: add diagram.

Upload using binary file

- The target receives the LUR file, listing the LUH files.
- The target requests the LUH files.
- The target reads and processes the LUH file and requests data files as listed in the binary load definition file.
- The target validates the LUH load checksum against the data files and content of the LUH file.

Upload using xml file

- The target receives the LUR file, listing the LUH files.
- The target requests the LUH files.
- The target requests the LUX files.
- The target reads and processes the LUX file and requests data files as listed in the XML load definition file.
- The target validates the LUX load checksum against the data files and content of the LUH file.

2.6.3 ARINC 615A Target Loader Implementation

2.0 OVERVIEW

Note: add diagram.

Upload using binary file

- The LUR file will list the binary load definition file (LUH).
- The target reads and processes the LUH file and requests data files as listed in the header file.
- The target validates the checksum against the load checksum listed in the binary LUH file.

Upload using XML file

- The LUR file will list the binary load definition file (LUH).
- The target reads the LUR file and requests the equivalent LUX file as read requests, switching the file extension from LUH to LUX.
- The loader serves the LUX files which are in the root part directory.
- The target reads and processes the LUX file and requests data files as listed in the XML file.

**ATTACHMENT 1
GENERIC DEFINITION OF PART DEFINITION FORMAT**

ATTACHMENT 1 GENERIC DEFINITION OF PART DEFINITION FORMAT

1.1 Syntax

The general file format structure is expressed using modified BNF (Backus-Naur Form) notation. It is standard industry practice to describe simple file formats using this formal notation.

Type	Description
Element	SEQUENCE element, or LIST0 element, or LIST1 element, or SIMPLE element, or BOOLEAN element
SEQUENCE	sequence of elements
SECTIONLIST	sequence of sections
LIST0 element	list of 0 or more elements
LIST1 element	list of 1 or more elements
SIMPLE element	STRING255 element, or INT64 element, or UINT64 element, or INT32 element, or UINT32 element, or BOOLEAN element
STR255	sequence of 0 to 255 characters
STR64K	sequence of 0 to 64535 characters
SINT32	signed Integer number, 32 bits
UINT32	unsigned integer number, 32 bits
SINT64	signed Integer number, 64 bits
UINT64	unsigned integer number, 64 bits
BOOLEAN	boolean value: 0 or 1

1.1.1 Load Definition File Sections

```
ldf-file ::=                               SEQUENCE (
    <opening>
    <file-format-version>
    <ldf-sections>
)
```

```
file-format-version ::=                   STR255
```

```
ldf-sections ::=                          SECTIONLIST(
    <load-description>
    <thw-definitions>
```

**ATTACHMENT 1
GENERIC DEFINITION OF PART DEFINITION FORMAT**

```

    <file-definitions>
    <load-integrity-definition>
)

```

1.1.2 Load Description Section

```

load-description ::=                               SEQUENCE (
    <load-partflags>
    <load-partnumber>
    <load-type-description>
    <load-type-id>
)

```

```

load-partflags ::=                                UINT32

```

```

load-partnumber ::=                               STR255

```

```

load-type-description ::=                        STR255

```

```

load-type-id ::=                                 STR255

```

1.1.3 Target Definitions Section

```

thw-definitions ::=                              LIST1 (
    <thw_definition>
)

```

```

thw-definition ::=                              SEQUENCE (
    <thw-id>
    <thw-positions>
)

```

```

thw-id ::=                                       STR255

```

```

thw-positions ::=                               LIST0 (
    <thw_positions>
)

```

```

thw-position ::=                                STR255

```

1.2 File Definitions Section

```

file-definitions ::=                             LIST1 (
    <file-definition>
)

```

```

file-definition ::=                             SEQUENCE(
    <file-loadable>
    <file-name>
    <file-location>
    <file-number>
    <file-description>
    <file-length>
    <file-integrity-definition>
)

```

**ATTACHMENT 1
GENERIC DEFINITION OF PART DEFINITION FORMAT**

```

)
file-loadable ::=          BOOLEAN
file-name ::=             STR255
file-location ::=        STR255
file-number ::=          STR255
file-description ::=     STR255
file-size ::=            UINT32
    
```

1.2.1 Integrity Definitions Section

```

file-integrity-definition ::= SEQUENCE(
    <integrity-type>
    <integrity-value>
)
load-integrity-definition ::= SEQUENCE(
    <integrity-type>
    <integrity-value>
)

integrity-type ::=          INT32
integrity-value ::=        STR255
    
```

1.3 File naming and locations

Binary Header File

The binary part description file shall reside in the part root directory.

The binary part description file shall have the file name MMMCCSSSSSSSSSS.LUH or MMM-CC-SSSS-SSSS.LUH.

Note: extract file naming convention from ARINC 665-3.

XML Header File

The XML part description file shall reside in the part root directory.

The SML part description file shall have the file name MMMCCSSSSSSSSSS.LUX or MMM-CC-SSSS-SSSS.LUX.

The XML file shall have all the same information fields as the binary file, in XML notation, listed in the same order.

Data Files

All files of the load shall reside in the root part directory or a subdirectory of the root part directory.

ATTACHMENT 1
GENERIC DEFINITION OF PART DEFINITION FORMAT

1.4 Semantic

Field Name	Data Type	Definition
file-format-version	UINT32	This field shall be set to 0x8042. The field identifies the file format version associated with this report. Other values are invalid.
load-partflags	UINT32	Bit 0 of this field shall be set to 1 if the part is uploadable (from data loader to target). Bit 0 of this field shall be set to 0 if the part is not uploadable (from data loader to target). Bit 1 of this field shall be set to 1 if the part is downloadable (from target to dataloader). Bit 1 of this field shall be set to 0 if the part is not downloadable (from target to dataloader). Bit 2 to Bit 31 shall be set to 0. Bit 2 to bit 31 are reserved for future use. If the field is not present (optional), the load shall be uploadable and not downloadable. <i>Note: this field is currently under review for removal by SDL.</i>
load-partnumber	STR64K	This field shall contain the load part number of the load. The format shall be MMM-CC-SSSS-SSSS or MMMCCSSSSSSSSSS. <i>Note: definition to be expanded based on definition from ARINC 665-3.</i>
load-type-description	STR64K	This field shall contain the Load Type Description for the load. <i>Note: definition to be expanded based on definition from ARINC 665-3.</i>
load-type-id	STR64K	This field shall contain the Load Type ID for the load. <i>Note: definition to be expanded based on definition from ARINC 665-3.</i>
thw-id	STR64K	This field shall be a THW_ID of a target computer applicable for the load. The load definition file shall list all applicable THW_IDs for the load.
thw_position	STR64K	This field shall be a POSITION for a valid THW_ID applicable for the load. The load definition file shall list all POSITIONS applicable for applicable THW_IDs applicable for the load.
file-loadable	UINT32	This field shall be set to 1 if the file as specified in field “file-name” should be requested by the target LRU as part of the upload. This field shall be set to 0 if the file as specified in field “file-name” should not be requested by the target LRU as part of the upload. <i>Note: The field “file-loadable” identifies if an individual file within an uploadable load should be transferred to the target LRU. Files declared with value “file-loadable” set to zero are files residing inside the load, but not being loaded to the target. The load checksum does not cover files set to “file-loadable”=0.</i>

**ATTACHMENT 1
GENERIC DEFINITION OF PART DEFINITION FORMAT**

file-name	STR64K	This field shall contain the file name of the file. The file name excludes directory, location, or path information inside the load. <i>Note: add definition of what are valid file names based on definition by 665-3.</i>
file-number	STR64K	If the field exists, it shall contain a printable part number string of the file associated with the field "file-name." If there is no printable part number string of the file associated with the field "file-name," this field shall be set to an empty string, or not exist (optional). The file part number is at the discretion of the manufacturer of the load. This field can be used to support short loading. <i>Note: this field is currently under review for removal by SDL.</i>
file-location	STR64K	If the file associated with the field "file-name" resides within the part root directory, this field shall either be an empty string, or not exist (optional). If the file "file-name" does not reside within the part root directory, this field shall be set to the relative path from the part root directory to the subdirectory of the file associated with the field "file-name." The file associated with "file-name" shall reside in the part root directory, or in subdirectories of the part root directory.
file-description	STR64K	If the field exists, it shall contain a printable description string of the file associated with the field "file-name." If there is no printable description string of the file associated with the field "file-name," this field shall be set to an empty string, or not exist (optional).
file-size	UINT32	This field shall contain the number of bytes of the file associated with the field "file-name."
file-integrity-type	UINT32	This field shall identify the integrity type algorithm used to compute the checksum of the file associated with the field "file-name." This field shall not exist (optional) if the load does not provide the checksum of the file. This field shall not exist (optional) if the field "file-uploadable" associated with the field "file-name" is set to 0. This field shall not exist (optional) if the field "file-uploadable" associated with the field "file-name" does not exist. <i>Note: Specify values for the algorithms SHA-1, CRC32, CRC64, as defined in ARINC 665-3.</i>
file-integrity-value	STR64K	This field shall identify the computed integrity type value used to compute the checksum of the file associated with the field "file-name," based on the integrity type algorithm specified in field "file-integrity-type." This field shall not exist (optional) if the field "file-integrity-type" associated with the field "file-name" does not exist."
load-integrity-type	UINT32	This field shall identify the integrity type algorithm used to compute the checksum of the load, including all files declared as "file-uploadable." This field shall be computed as the checksum spanning all files declared in the binary header file as "file-uploadable" and the content of the binary load description file up to

**ATTACHMENT 1
 GENERIC DEFINITION OF PART DEFINITION FORMAT**

		the field "load-integrity-type." <i>Note: Specify values for the algorithms SHA-1, CRC32, CRC64, as defined in ARINC 665-3.</i>
load-integrity-value	STR64K	This field shall identify the computed integrity type value used to compute the checksum of the load, including all files declared as "file-uploadable."

**ATTACHMENT 2
XML FORMAT**

ATTACHMENT 2 XML FORMAT

2.1 XML Template

2.1.1 Load Description File Sections

2.1.1.1 Opening Declaration

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema

    xmlns:xsd="http://www.w3.org/2001/XMLSchema"

    elementFormDefault="qualified"

    attributeFormDefault="unqualified"

>
```

2.1.1.2 File Format Version

```
< <xsd:element name="file-format-version">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:element>
```

2.1.1.3 LDF sections

```
<xsd:element name="ldf-sections">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="load-description"/>
            <xsd:element ref="thw-definitions"/>
            <xsd:element ref="file-definitions"/>
            <xsd:element ref="load-integrity-definition"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

2.2 Load Description Section

```
<xsd:element name="load-description">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="load-partflags"/>
            <xsd:element ref="load-partnumber"/>
            <xsd:element ref="load-type-description"/>
            <xsd:element ref="load-type-id"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="load-partflags">
```

ATTACHMENT 2
XML FORMAT

```

    <xsd:simpleType>
      <xsd:restriction base="xsd:integer"/>
    </xsd:simpleType>
  </xsd:element>

```

```

<xsd:element name="load-partnumber">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>

```

```

<xsd:element name="load-type-description">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>

```

```

<xsd:element name="load-type-id">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>

```

2.2.1 Target Definitions Section

```

<xsd:element name="thw-definitions">
  <xsd:complexType>
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element ref="thw-definition"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

<xsd:element name="thw-definition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="thw-id"/>
      <xsd:element ref="thw-positions"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

<xsd:element name="thw-id">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>

```

```

<xsd:element name="thw-positions">
  <xsd:complexType>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="thw-position"/>
    </xsd:sequence>
  </xsd:complexType>

```

**ATTACHMENT 2
XML FORMAT**

```
</xsd:element>
```

```
<xsd:element name="thw-position">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>
```

2.2.2 File Definitions Section

```
<xsd:element name="file-definitions">
  <xsd:complexType>
    <xsd:sequence maxOccurs="unbounded">
      <xsd:element ref="file-definition"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="file-definition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="file-loadable"/>
      <xsd:element ref="file-name"/>
      <xsd:element ref="file-location"/>
      <xsd:element ref="file-number"/>
      <xsd:element ref="file-description"/>
      <xsd:element ref="file-size"/>
      <xsd:element ref="file-integrity-definition"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="file-loadable" type="xsd:boolean"/>
```

```
<xsd:element name="file-name">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="file-location">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="file-number">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>
```

```
<xsd:element name="file-description">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>
```

ATTACHMENT 2
XML FORMAT

```

<xsd:element name="file-size">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer"/>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="file-integrity-definition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="integrity-type"/>
      <xsd:element ref="integrity-value"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

2.2.3 Integrity Definitions Section

```

<xsd:element name="load-integrity-definition">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="integrity-type"/>
      <xsd:element ref="integrity-value"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="integrity-type">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer"/>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="integrity-value">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
</xsd:element>

```

2.2.3.1 Closing Declaration

```
</xsd:schema>
```

2.3 Example XML Load Definition File

```

<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema

xmlns:ldf="http://www.acme.org/ldf.xsd">
  <file-format-version>8042</file-format-version>
  <ldf-sections>
    <load-description>
      <load-partflags>1</load-partflags>
      <load-partnumber>ACM-12-1234-1235</load-partnumber>
      <load-type-description>roadrunner</load-type-description>
      <load-type-id>42</load-type-id>
    </load-description>
  </ldf-sections>
</xsd:schema>

```

**ATTACHMENT 2
XML FORMAT**

```

</load-descriptions>
<thw-definitions>
  <thw_definition>
    <thw-id>ACME1</thw_id>
    <thw-positions>
      <thw_position>L</thw_position>
      <thw_position>F</thw_position>
    </thw-positions>
  </thw_definition>
  <thw_definition>
    <thw-id>ACME2</thw_id>
    <thw-positions>
      <thw_position>1</thw_position>
    </thw-positions>
  </thw_definition>
  <thw_definition>
    <thw-id>ACME2</thw_id>
    <thw-positions>
      <thw_position></thw_position>
    </thw-positions>
  </thw_definition>
</thw-definitions>
<file-definitions>
  <file-definition>
    <file-loadable>1</file-loadable>
    <file-name>Sylvester1.doc</file-name>
    <file-location>subdir1/anothersubdir</file-location>
    <file-number>1</file-number>
    <file-description>a file</file-description>
    <file-length>22</file-length>
  </file-definition>
  <file-definition>
    <file-loadable>1</file-loadable>
    <file-name>Sylvester2.doc</file-name>
    <file-location>subdir1</file-location>
    <file-number>1</file-number>
    <file-description>another file</file-description>
    <file-length>42</file-length>
  </file-definition>
</file-definitions>
<integrity-definitions>
  <ldf-integrity-definition>
    <integrity-type>1</integrity-type>
    <integrity-value>98765432</integrity-value>
  </ldf-integrity-definition>
  <part-integrity-definition>
    <integrity-type>1</integrity-type>
    <integrity-value>1234567</integrity-value>
  </part-integrity-definition>
</integrity-definitions>
</ldf-sections>

```

**ATTACHMENT 3
BINARY FORMAT**

ATTACHMENT 3 BINARY FORMAT

3.1 Template for Binary File

Type	Description
Element	SEQUENCE element, or LIST0 element, or LIST1 element, or SIMPLE element, or BOOLEAN element
SEQUENCE	Sequence of elements, written to the file in the order of occurrence. No padding or additional alignment between fields.
SECTIONLIST	<ul style="list-style-type: none"> • sequence of relative pointers (pointing to first byte of section • followed by sequence of sections. <p>No padding or additional alignment between pointers and sections.</p> <p>The relative pointers are of type RPTR32.</p>
LIST0 element	<ul style="list-style-type: none"> • Length of list (number of elements) as UINT32, • Followed by list of relative pointers/elements • At the beginning of each list element a relative pointer is created to the next element. The relative pointers are of type RPTR32. The last list element has a pointer value of 0. <p>No padding or additional alignment between pointers and elements.</p> <p>In case of 0 elements, the length of the list has the value 0, followed by a zero pointer of the first block.</p>
LIST1 element	Same as list 0, except list contains at least 1 element.
SIMPLE element	Simple element. No padding or additional alignment between elements.
STR64K	2-byte string length, followed by sequence of 0 to

**ATTACHMENT 3
BINARY FORMAT**

	64535 characters, terminated by 0 character.
SINT32	signed Integer number, 32 bits
UINT32	unsigned integer number, 32 bits
SINT64	signed Integer number, 64 bits
UINT64	unsigned integer number, 64 bits
BOOLEAN	unsigned integer, 32 bits. 0: TRUE, 1: FALSE, other values invalid.

Fields of Binary File Format

Note: This section needs update to be in sync with grammar/templates.

3.1.1 Load Definition File Sections

Type	Length	field-name	Definition
none	0	opening	No opening for binary file
UINT32	4	file-format-version	4 bytes file format version
RPTR32	4	relptr-load-description	4 byte relative pointer to first byte of section
RPTR32	4	relptr-target-definitions	4 byte relative pointer to first byte of section
RPTR32	4	relptr-file-definitions	4 byte relative pointer to first byte of section
RPTR32	4	relptr-integrity-definitions	4 byte relative pointer to first byte of section
SECTION LOAD DESCRIPTION			
SECTION TARGET DEFINITIONS			
SECTION FILE DEFINITIONS			
SECTION INTEGRITY DEFINITIONS			

ATTACHMENT 3
BINARY FORMAT

3.1.2 Load Description Section

Type	Length	field-name	Value
LOAD DESCRIPTION SECTION			
UINT32	4	Load-partflags	Bit 0 set: upload part Bit 1 set: download part
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	Load-partnumber	665 part number of load
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	Load-type-description	Load type description string
UINT32	4	Load-type-id	Load type ID

3.1.3 Target Definitions Section

Type	Length	field-name	Value
SECTION THW-DEFINITIONS			
LIST1	0	thw-definitions	
UINT32	4	length thw-definitions	Number of THW sequences
RPTR32	4	relptr thw-definition	Pointer to next THW sequence
SEQUENCE	0	thw_definition	
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	thw-id	Target hardware ID string
LIST0	0	thw-positions	
UINT32	even	length thw-positions	Number of thw-positions

**ATTACHMENT 3
BINARY FORMAT**

Type	Length	field-name	Value
RPTR32	4	relptr thw-position	Pointer to next thw-position
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	thw-position	Position string

3.2 File Definitions Section

Type	Length	field-name	Value
SECTION FILE DEFINITIONS			
LIST1	0	File-definitions	
UINT32	4	Length file-definitions	Number of file definitions
RPTR32	4	Relptr file-definition	Relative pointer to next file definition
SEQUENCE	0	file definition	
BOOLEAN	4	File-loadable	1: loadable 0: not loadable Other values are invalid
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-name	File name field
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-location	File location field
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-number	File number field
STR64K	2 + (string length + 1)	File-description	File description field

**ATTACHMENT 3
BINARY FORMAT**

Type	Length	field-name	Value
	+ (string length + 1) modulo 2		
UINT32	4	File-length	File length field

3.2.1 Integrity Definitions Section

Type	Length	field-name	Value
SECTION INTEGRITY DEFINITIONS			
SEQUENCE	0	Integrity-definitions	
SEQUENCE	0	LDF-integrity-definition	
UINT32	4	LDF-integrity-type	LDF integrity type
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	LDF-integrity-value	LDF integrity value
SEQUENCE	0	Part-integrity-definition	
UINT32	4	Part-integrity-type	Part integrity type
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	Part-integrity-value	Part integrity value

3.3 Example Binary Format

Note: This section needs update to be in sync with grammar/templates.

```

<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema

xmlns:ldf="http://www.acme.org/ldf.xsd">
<file-format-version>8042</file-format-version>
<ldf-sections>
...
</ldf-sections>
    
```

**ATTACHMENT 3
BINARY FORMAT**

Translates to :

Type	Length	field-name	Value
None	0	opening	
UINT32	4	file-format-version	8042
RPTR32	4	relptr-load-description	TBD
RPTR32	4	relptr-target-definitions	TBD
RPTR32	4	relptr-file-definitions	TBD
RPTR32	4	relptr-integrity-definitions	TBD
SECTION LOAD DESCRIPTION			
SECTION TARGET DEFINITIONS			
SECTION FILE DEFINITIONS			
SECTION INTEGRITY DEFINITIONS			

```

<load-description>
  <load-partflags>1</load-partflags>
  <load-partnumber>ACM-12-1234-1235</load-partnumber>
  <load-type-description>roadrunner</load-type-description>
  <load-type-id>42</load-type-id>
</load-descriptions>
    
```

Translates to :

Type	Length	field-name	Value
LOAD DESCRIPTION SECTION			
UINT32	4	Load-partflags	1
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	Load-partnumber	“ACM-12-1234-1235”
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	Load-type-description	“roadrunner”

ATTACHMENT 3
BINARY FORMAT

Type	Length	field-name	Value
UINT32	4	Load-type-id	42

```

<thw-definitions>
  <thw_definition>
    <thw-id>ACME1</thw_id>
    <thw-positions>
      <thw_position>L</thw_position>
      <thw_position>R</thw_position>
    </thw-positions>
  </thw_definition>
  <thw_definition>
    <thw-id>ACME2</thw_id>
    <thw-positions>
      <thw_position>1</thw_position>
    </thw-positions>
  </thw_definition>
  <thw_definition>
    <thw-id>ACME3</thw_id>
    <thw-positions>
      <thw_position></thw_position>
    </thw-positions>
  </thw_definition>
</thw-definitions>
  
```

Translates to:

Type	Length	field-name	Value
SECTION THW-DEFINITIONS			
LIST1	0	thw-definitions	
UINT32	4	length thw-definitions	3
RPTR32	4	relptr thw-definition	TBD
SEQUENCE	0	thw_definition	
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	thw-id	“ACME1”
LIST0	0	thw-positions	
UINT32	even	length thw-positions	2
RPTR32	4	relptr thw-position	TBD

**ATTACHMENT 3
BINARY FORMAT**

Type	Length	field-name	Value
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	thw-position	“L”
RPTR32	4	relptr thw-position	TBD
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	thw-position	“R”
RPTR32	4	relptr thw-definition	TBD
SEQUENCE	0	thw_definition	
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	thw-id	“ACME2”
LIST0	0	thw-positions	
UINT32	even	length thw-positions	1
RPTR32	4	relptr thw-position	TBD
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	thw-position	“1”
RPTR32	4	relptr thw-definition	TBD
SEQUENCE	0	thw_definition	
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	thw-id	“ACME3”
LIST0	0	thw-positions	
UINT32	even	length thw-positions	0
RPTR32	4	relptr thw-position	0

**ATTACHMENT 3
BINARY FORMAT**

```

<file-definitions>
  <file-definition>
    <file-loadable>1</file-loadable>
    <file-name>Sylvester1.doc</file-name>
    <file-location>subdir1/anothersubdir</file-location>
    <file-number>A1</file-number>
    <file-description>a file</file-description>
    <file-length>22</file-length>
  </file-definition>
  <file-definition>
    <file-loadable>1</file-loadable>
    <file-name>Sylvester2.doc</file-name>
    <file-location>subdir1</file-location>
    <file-number>A2</file-number>
    <file-description>another file</file-description>
    <file-length>42</file-length>
  </file-definition>
</file-definitions>
    
```

Translates to:

Type	Length	field-name	Value
SECTION FILE DEFINITIONS			
LIST1	0	File-definitions	
UINT32	4	Length file-definitions	2
RPTR32	4	Relptr file-definition	TBD
SEQUENCE	0	file definition	
BOOLEAN	4	File-loadable	1
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-name	"Sylvester1.doc"
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-location	"subdir1"
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-number	"A1"

ATTACHMENT 3
BINARY FORMAT

Type	Length	field-name	Value
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-description	"a file"
UINT32	4	File-length	22
RPTR32	4	Relptr file-definition	TBD
SEQUENCE	0	file definition	
BOOLEAN	4	File-loadable	1
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-name	"Sylvester2.doc"
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-location	"subdir1"
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-number	"A2"
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	File-description	"another file"
UINT32	4	File-length	42

```

<integrity-definitions>
  <ldf-integrity-definition>
    <integrity-type>1</integrity-type>
    <integrity-value>98765432</integrity-value>
  </ldf-integrity-definition>
  <part-integrity-definition>
    <integrity-type>1</integrity-type>
    <integrity-value>1234567</integrity-value>
  </part-integrity-definition>
</integrity-definitions>

```

**ATTACHMENT 3
BINARY FORMAT**

Translates to:

Type	Length	field-name	Value
SECTION INTEGRITY DEFINITIONS			
SEQUENCE	0	Integrity-definitions	
SEQUENCE	0	LDF-integrity-definition	
UINT32	4	LDF-integrity-type	1
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	LDF-integrity-value	98765432
SEQUENCE	0	Part-integrity-definition	
UINT32	4	Part-integrity-type	1
STR64K	2 + (string length + 1) + (string length + 1) modulo 2	Part-integrity-value	1234567

**APPENDIX A
LIST OF ACRONYMS**

APPENDIX A LIST OF ACRONYMS

< Insert acronyms and glossary >

**APPENDIX B
GLOSSARY**

APPENDIX B GLOSSARY

< Insert applicable XML conventions, following approach taken in ARINC 827 >

APPENDIX C
XML CONVENTIONS

APPENDIX C XML CONVENTIONS

APPENDIX D
XML SCHEMA

APPENDIX D XML SCHEMA

```

<?xml version="1.0" encoding="UTF-8"?>
<!--2009-11-02-IV carsten schweigert -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xsd:element name="ldf-file">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="file-format-version"/>
        <xsd:element ref="ldf-sections"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="file-format-version">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="ldf-sections">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="load-description"/>
        <xsd:element ref="thw-definitions"/>
        <xsd:element ref="file-definitions"/>
        <xsd:element ref="load-integrity-definition"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="load-description">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="load-partflags"/>
        <xsd:element ref="load-partnumber"/>
        <xsd:element ref="load-type-description"/>
        <xsd:element ref="load-type-id"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="load-partflags">
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer"/>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="load-partnumber">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="load-type-description">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="load-type-id">
    <xsd:simpleType>

```

**APPENDIX D
XML SCHEMA**

```

        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="thw-definitions">
    <xsd:complexType>
        <xsd:sequence maxOccurs="unbounded">
            <xsd:element ref="thw-definition"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="thw-definition">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="thw-id"/>
            <xsd:element ref="thw-positions"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="thw-id">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="thw-positions">
    <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="thw-position"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="thw-position">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="file-definitions">
    <xsd:complexType>
        <xsd:sequence maxOccurs="unbounded">
            <xsd:element ref="file-definition"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="file-definition">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="file-loadable"/>
            <xsd:element ref="file-name"/>
            <xsd:element ref="file-location"/>
            <xsd:element ref="file-number"/>
            <xsd:element ref="file-description"/>
            <xsd:element ref="file-size"/>
            <xsd:element ref="file-integrity-definition"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="file-loadable" type="xsd:boolean"/>
<xsd:element name="file-name">

```

APPENDIX D
XML SCHEMA

```

        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="file-location">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="file-number">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="file-description">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="file-size">
        <xsd:simpleType>
            <xsd:restriction base="xsd:integer"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="file-integrity-definition">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="integrity-type"/>
                <xsd:element ref="integrity-value"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="load-integrity-definition">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="integrity-type"/>
                <xsd:element ref="integrity-value"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="integrity-type">
        <xsd:simpleType>
            <xsd:restriction base="xsd:integer"/>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="integrity-value">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
    </xsd:element>
</xsd:schema>

```

**APPENDIX E
XML EXAMPLE – SMALL FILE**

APPENDIX E XML EXAMPLE – SMALL FILE

```

<?xml version="1.0" encoding="UTF-8"?>
<ldf-file xsi:noNamespaceSchemaLocation="schema_ldf-2009-11-02.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <file-format-version>String</file-format-version>
  <ldf-sections>
    <load-description>
      <load-partflags>0</load-partflags>
      <load-partnumber>String</load-partnumber>
      <load-type-description>String</load-type-description>
      <load-type-id>String</load-type-id>
    </load-description>
    <thw-definitions>
      <thw-definition>
        <thw-id>String</thw-id>
        <thw-positions/>
      </thw-definition>
    </thw-definitions>
    <file-definitions>
      <file-definition>
        <file-loadable>true</file-loadable>
        <file-name>String</file-name>
        <file-location>String</file-location>
        <file-number>String</file-number>
        <file-description>String</file-description>
        <file-size>0</file-size>
        <file-integrity-definition>
          <integrity-type>0</integrity-type>
          <integrity-value>String</integrity-value>
        </file-integrity-definition>
      </file-definition>
    </file-definitions>
    <load-integrity-definition>
      <integrity-type>0</integrity-type>
      <integrity-value>String</integrity-value>
    </load-integrity-definition>
  </ldf-sections>
</ldf-file>

```

**APPENDIX F
XML EXAMPLE – COMPLETE FILE**

APPENDIX F XML EXAMPLE – COMPLETE FILE

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<ldf-file xsi:noNamespaceSchemaLocation="schema_ldf-2009-11-02.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <file-format-version>String</file-format-version>
  <ldf-sections>
    <load-description>
      <load-partflags>1</load-partflags>
      <load-partnumber>String</load-partnumber>
      <load-type-description>String</load-type-description>
      <load-type-id>String</load-type-id>
    </load-description>
    <thw-definitions>
      <thw-definition>
        <thw-id>String</thw-id>
        <thw-positions>
          <thw-position>String</thw-position>
          <thw-position>String</thw-position>
        </thw-positions>
      </thw-definition>
      <thw-definition>
        <thw-id>String</thw-id>
        <thw-positions>
          <thw-position>String</thw-position>
          <thw-position>String</thw-position>
        </thw-positions>
      </thw-definition>
    </thw-definitions>
    <file-definitions>
      <file-definition>
        <file-loadable>true</file-loadable>
        <file-name>String</file-name>
        <file-location>String</file-location>
        <file-number>String</file-number>
        <file-description>String</file-description>
        <file-size>0</file-size>
        <file-integrity-definition>
          <integrity-type>0</integrity-type>
          <integrity-value>String</integrity-value>
        </file-integrity-definition>
      </file-definition>
      <file-definition>
        <file-loadable>true</file-loadable>
        <file-name>String</file-name>
        <file-location>String</file-location>
        <file-number>String</file-number>
        <file-description>String</file-description>
        <file-size>0</file-size>
        <file-integrity-definition>
          <integrity-type>0</integrity-type>
          <integrity-value>String</integrity-value>
        </file-integrity-definition>
      </file-definition>
    </file-definitions>
  </ldf-sections>
</ldf-file>

```

**APPENDIX F
XML EXAMPLE – COMPLETE FILE**

```
<load-integrity-definition>  
  <integrity-type>0</integrity-type>  
  <integrity-value>String</integrity-value>  
</load-integrity-definition>  
</ldf-sections>  
</ldf-file>
```

**APPENDIX G
BINARY EXAMPLE – SMALL FILE**

APPENDIX G BINARY EXAMPLE – SMALL FILE

TBD

**APPENDIX H
BINARY EXAMPLE – COMPLETE FILE**

APPENDIX H BINARY EXAMPLE – COMPLETE FILE

TBC

**APPENDIX I
COMPARISON TO OTHER SOFTWARE STANDARDS**

APPENDIX I COMPARISON TO OTHER SOFTWARE STANDARDS

ARINC 665 Advantages

- Industry-wide standard
- Implemented in thousands of parts generated
- Standard matured since few years

ARINC 665 Disadvantages

- Regular need for clarifications as part of ARINC meetings
- Ambiguity still exists, very high integration costs for manufacturers of hundreds of LRUs
- No formal languages used to express syntax
- No formal languages used to express semantics
- Complex header file format
- Protocol not humanly readable, need for special tools

Intended Benefit of XML software part

- Definition of simple, standardized ARINC 665-X part header
- Support of loader needs and electronic distribution
- Usage XML-compliant subset to ease readability and electronic processing by COTS tools and methods
- Support of tools and electronic distribution
 - Definition of certifiable sub-set of XML notation that can be expressed by simple grammar
- Support of target needs
 - Definition of subset that supports processing by finite state machine, simple top-down or sequential parser
 - Formal notation, simple structure allows translation of text representation of XML format to binary format.

Relationship to EDS Crates

- Load structure transparent to crate

**APPENDIX I
COMPARISON TO OTHER SOFTWARE STANDARDS**

- XML format for both crate definition file and load definition file

Relationship to SDL Media

- ARINC 665 contains definitions for part definition files, batch files, and transport media definition files.
- XML software part definition replaces 665 binary part definition (“header”) by simplified and formally defined version (binary and text format).
- XML software part definition could also replace transport media definition files by simplified and formally defined versions. Not in current scope of this report.
- XML software part definition could also replace 665 batch definition file by simplified and formally defined versions. Not in current scope of this report.