AEEC

Together "We Set the Standard."

To	SDL Subcommittee	**Date**	May 15, 2015
From	Scott L. Smith smitty@sae-itc.org tel +1 240-334-2582	**Reference**	15-059/SDL-095

Subject **Draft Circulation**
Draft 2 of Supplement 4 to ARINC Report 665: *Loadable Software Standards*

Summary The ARINC Industry Activities staff prepared this draft in response to ARINC Project Initiation/Modification (APIM) 15-003. This draft includes a number of technical changes supported by the software data loader community as follows:

- Manufacturer's Codes and Assignment
- Software Load File naming
- Header File Definition
- Rules for CRC Calculation
- Other technical changes per Errata inputs

Additionally, editorial changes have been made where necessary.

All technical changes are shown in **blue bold**. Text that has been deleted may or may not be shown in ~~strikethrough~~ depending on the extent of the deletions.

Action The Software Data Loader (SDL) Subcommittee will review this draft during its next meeting, tentatively scheduled for June 16-19, 2015, in Copenhagen, Denmark. For details of this meeting, refer to the SDL Meeting Announcement posted on the ARINC website at the following URL:

http://www.aviation-ia.com/events/

If you wish your comments to be considered at the next meeting, please respond in writing (e-mail) to Scott Smith by June 12, 2015.

cc SAI Subcommittee

Preamble:

When this Supplement has been completed, adopted and published, Sections A, B and C will be affixed to the end of the published Specification. These pages, currently numbered a, b, c…, are used to explain the changes that will be made by this draft Supplement. The content of Sections A, B and C is under development in parallel with the changes to the body of the existing standard. Therefore, changes to their content are shown in blue bold in the same manner as changes to the body of the document.

Section A is written as it is expected to read when the Supplement is mature.

When the changes developed in this Supplement are integrated into the existing standard, they will be identified by blue bold.

Section C contains a cumulative list of entries describing the changes to be incorporated by this Supplement.

DRAFT 2 OF
SUPPLEMENT 4 TO
ARINC REPORT 665
LOADABLE SOFTWARE STANDARDS

Published: TBD

Prepared by the AEEC

Report 665-4 Adopted by the AEEC Executive Committee: TBD

A. PURPOSE OF THIS DOCUMENT

This supplement provides corrections or updates to air transport aircraft loadable software requirements, including:

- Part number conventions
- Nomenclature considerations
- Security and integrity checking
- Manufacturer's Code Request processes

The term "software transport media" has been changed throughout the document to "media set parts" to reflect current and accurate terminology.

In addition, editorial changes include updates to ARINC Industry Activities web, email, and phone addresses. Material from collected errata and working papers have been incorporated into the document as well.

B. ORGANIZATION OF THIS SUPPLEMENT

In this document **blue bold** text is used to indicate those areas of text changed by the current supplement only.

C. CHANGES TO ARINC REPORT 665 INTRODUCED BY THIS SUPPLEMENT

This section presents a complete listing of the changes to the document introduced by this supplement. Each change is identified by the section number and the title as it will appear in the complete document. Where necessary, a brief description of the change is included.

1.2 Applicability

Text added to reference **ARINC Specification 838:** *Loadable Software Part Definition* as an XML option for air transport software.

2.1.2 Manufacturer's Codes Assignment

Text modified to correct ARINC Industry Activities' web address, and to update the methods of requesting an MMM code.

2.1.3 Check Characters in the Software PN

Text modified referencing document section defining how check characters are derived.

2.2.1 Software Load Structure

Text added to denote the differences between Loadable Software Parts (LSPs) and Media Set Parts (MSPs). Guidance for directory structures is also added.

2.2.2 Software Load File Naming

Text updated defining unallowable characters in filenames.

2.2.3.1 Header File Content and Format

Table 2.2.3-1, Header File Content, modified providing guidance on word length and associated pointer fields.

2.2.3.1.10 Pointer to Number of Target HW ID with Positions

Text added providing guidance on cases where the field is set to zero and subsequently omitted.

2.2.3.1.18 Load Type ID

Text added recommending that this field should be the same as the intended target.

2.2.3.1.36 Data File PN

Text added recommending that this field is unique within a given LSP.

Commentary added explaining how the data file part number is used.

2.2.3.1.48 Support File PN Length

Text added providing guidance on cases where the field is set to zero and subsequently omitted.

2.2.3.1.49 Support File PN

Text added recommending that this field is unique within a given LSP.

2.2.3.1.61 Load Check Value

Text modified to clarify the check value type, and on cases where the field is set to zero and subsequently omitted.

2.2.3.1.63 Load CRC

Text added defining the order and the steps to calculate the 32-bit Load CRC.

2.2.3.2 Data File Content and Format

Text added identifying the LSP supplier as the originator of the data file format.

2.3.1 Batch File

Text added providing guidance on the creation and usage of batch file parts.

Table 2.3.1-1, Batch File Content, modified providing guidance on word length and associated pointer fields.

2.3.1.8 Batch File PN

Text added recommending that this field is unique from any LSP part numbers.

2.3.1.14 Target HW ID POS

Text added recommending that this field should be the same as the intended target, as well as consistent with all subordinate target hardware values.

2.3.1.17 Header File Name

Text added recommending that this field should be the same as the Load PN.

3.0 Loadable Media Set Parts

Title of Section 3 changed to reflect current and accurate terminology.

4.3.1 8-Bit CRC

Text added describing the 8-Bit CRC calculation process and results.

4.3.2 16-Bit CRC

Text added describing the 16-Bit CRC calculation process and results.

3.2.4.1.1 Storage of ARINC 615 Parts

Reference to ARINC 641 added.

4.2 Rules for CRC Calculation

Text added prescribing the order of operations for computing CRC values.

4.3.1 8-Bit CRC

Text added describing the 8-Bit CRC calculation process and results.

4.3.2 16-Bit CRC

Text added describing the 16-Bit CRC calculation process and results

5.1 Integrity Check Methods

Text added defining the order of operations for calculating check values.

5.2 Data Check Value Enumeration

Text added providing guidance for 8-Bit CRC checks used only for part number integrity.

Attachment 1 Manufacturer's Code Assignments

Text modified to reference ARINC Industry Activities web addresses.

Appendix C File Formats

Table of definitions modified with correct field titles.

Appendix I Reference Guide

Table of air transport software standards updates.

Appendix J Airplane Loadable Software – Request for Manufacturer's Code Designation

Text modified to reference ARINC Industry Activities web and fax addresses.

Appendix M Considerations for Implementing Supplement 2 to ARINC Report 665

This appendix was added from material originally published as a working paper. The material provides guidance for a subset of software developments that conform to ARINC Report 665-2. Minor differences between Supplements 1, 2, and 3 needed clarifications and the working paper provided these.

By adding Appendix M, the working paper is given higher visibility to software developers and implementers.

SAE Industry Technologies Consortia (SAE-ITC)
16701 Melford Blvd., Suite 120
Bowie, Maryland 20715 USA

DRAFT 2 OF SUPPLEMENT 4 TO ARINC REPORT 665 LOADABLE SOFTWARE STANDARDS

This draft dated: May 15, 2015

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Purpose

This document defines the aircraft industry's standards for Loadable Software Parts (LSPs) and ~~software transport Media~~ **Media Set Parts** (MSPs). It describes the common principles and rules to be applied to any part of a data load system to insure compatibility and inter-operability. It includes part numbering, content, labeling and formatting of an LSP, and a Media Set containing LSPs.

Loadable Software Airplane [Aircraft] Parts (LSAPs) are a subset of the LSP class of parts. All requirements for LSPs in this document also apply to LSAPs.

Uniform software LSP and ~~Media Set~~ formats enable suppliers to employ common (standardized) loadable software processes, procedures, and support tools.

It is intended that software loaders, tools, processes, and aircraft systems reference this standard for definition of Loadable Software Part and **Media Set Part** content and format. This should be independent of any specific data load system, production process, or aircraft system that uses the LSP.

1.2 Applicability

This standard is applicable to all Loadable Software Parts and **Media Set Parts** intended for use in aircraft programs, systems, equipment, and Line Replaceable Units (LRUs).

Alternately, an Extensible Markup Language (XML) based software standard for air transport aircraft may be used. ARINC Specification 838: *Loadable Software Part Definition* provides guidance on creating, identifying, and maintaining LSAPs using XML for human and machine readable needs.

1.3 Document Conventions

1.3.1 Terminology

This document is intended to assure interchangeability and interoperability between equipment independent of the manufacturer. The capabilities described in this document must be implemented to ensure a minimum level of compatibility between software loaders and tools, loadable software, and **Media Set Parts** designed to meeting the recommendations of this report.

In this document, "should" is used to define a capability that must be implemented for the unit to meet the minimum level of compatibility intended by this report. The terms "does," "is," and "will" are used to express a statement of fact based on other requirements. In this document, the term "may" is used to express an optional capability. Note in some cases, a capability "may" be implemented, but if it is, a specific aspect of it "should" be implemented in a specific manner. Otherwise, an incompatibility may exist with the aircraft or other interfacing equipment.

1.3.2 Field Formats

Data structures are represented by standard Hexadecimal and ASCII nomenclature. A data byte refers to a string of 8 bits, represented in the form of 0xFF, depicting a string of 8 bits of value (1). A data word refers to a string of 16 bits, represented in the form of 0xFFFF, where each F represents a string of 4 bits of value (1).

1.3.3 Data Type

Data fields should recognized as numeric data type, unless otherwise stipulated. Fields of textual content are identified as ASCII character strings. Selected fields may reflect either data type, based on specified options. Check value fields can be either numeric or character based on Check Value Type specifications.

1.4 File Format Evolution

This document defines standard file formats that enable software loaders, verifiers, electronic routers, and automated processes to accomplish their tasks on Loadable Software Parts (LSPs). It does so without prior knowledge of the supplier of the part, affiliated system, or aircraft model. One of the prime advantages of standardization is the cost saving of stable, long-lived tools for managing "standardized" parts. Long life requires flexibility to adapt as conditions change.

The specified file formats (and other standards) provide:

- The information necessary to support all anticipated needs.
- Maximum freedom for suppliers to control their own file content and format.
- The ability to evolve to meet unanticipated needs while maintaining maximum backward compatibility potential.
- Maximum backward compatibility with existing loadable software formats, loaders, tools, and aircraft systems (e.g., ARINC 615, ARINC 629, airline and supplier processes, etc.).

1.4.1 File Format Version Definition

Each file format definition includes a File Format Version Number field. This field indicates the specific version of the file format definition to which the file conforms.

Three classes of files are defined, with sufficient independence between them to allow independent evolution:

- Load files
- Batch files
- Media files

A specific File Format Version Number is associated with each class is assigned as follows:

- Load file format version: 0x8004
- Batch file format version: 0x9004
- Media file format version: 0xA004

To allow Load File format version 0x8004 to remain compatible with 0x8003 based loaders, the Load Check Values fields precede the Header File CRC and the Load CRC. Without specific pointers, these are recognized as the final 48 bits of the file.

With this design the Load Check Values fields are effectively appended to the User defined data, and will be so recognized by 0x8003 based loaders.

1.4.2 File Expansion Points

Expansion points are predefined positions in the file where new fields may be added in future versions of the file format.

Creators of LSPs and/or media sets should not insert fields of their own definition at any point in the file, except as overtly defined by formal updates to this report. Doing so will cause incompatibilities with tools and processes that depend on all files adhering to the “Loadable Software Standard.”

1.4.3 Support Tools and Loaders

Major downstream benefits may be achieved if interfacing tools and LSPs are designed with future file evolution in mind.

Using the defined “field pointers” to avoid “walking” through an expansion point will allow a tool to handle newer file format versions as if they were an earlier version. For example, a tool that understands version 1 files should be able to read a version 4 file as if it were a version 1 file.

When tools are updated to access fields added by later file versions, they should retain the ability to detect and deal appropriately with earlier file versions. This attribute is called backward compatibility.

1.4.4 Pointer Field Definition

The use of pointers in the ARINC 665 file listings (e.g., Header File) offers precise accounting and placement of all data components of the list. At any given point of reading, confirming, or transferring the part, a system can define its current point of reference. Use of pointers enable parts defined by early versions of ARINC 665 to be read by advanced systems, allowing backward compatibility.

ARINC Report 665-3 refined absolute pointer names to specifically indicate the field to which they point. However, their application did not change, pointing to the first, most significant bit of the same field.

Two types of pointers are defined as follows:

Absolute Pointer: number of 16-bit words from the beginning of the file to the field being pointed to (not including the first 16-bit word of the pointed field). For example, with actual definition of the Header Field format, the Pointer to Load PN Length field should have the unsigned integer value of 20 (0x0014).

Relative Pointer: number of 16-bit words between the relative pointer and the first 16-bit word of the pointed-to field. The relative pointer is included in the count, while the first word of the pointed-to field is not included in the count. In the example in Figure 1.4.4, the relative pointer value is 5, consisting of one 16-bit word (relative pointer), plus four 16-bit words (intermediate fields). The value is an unsigned integer value.

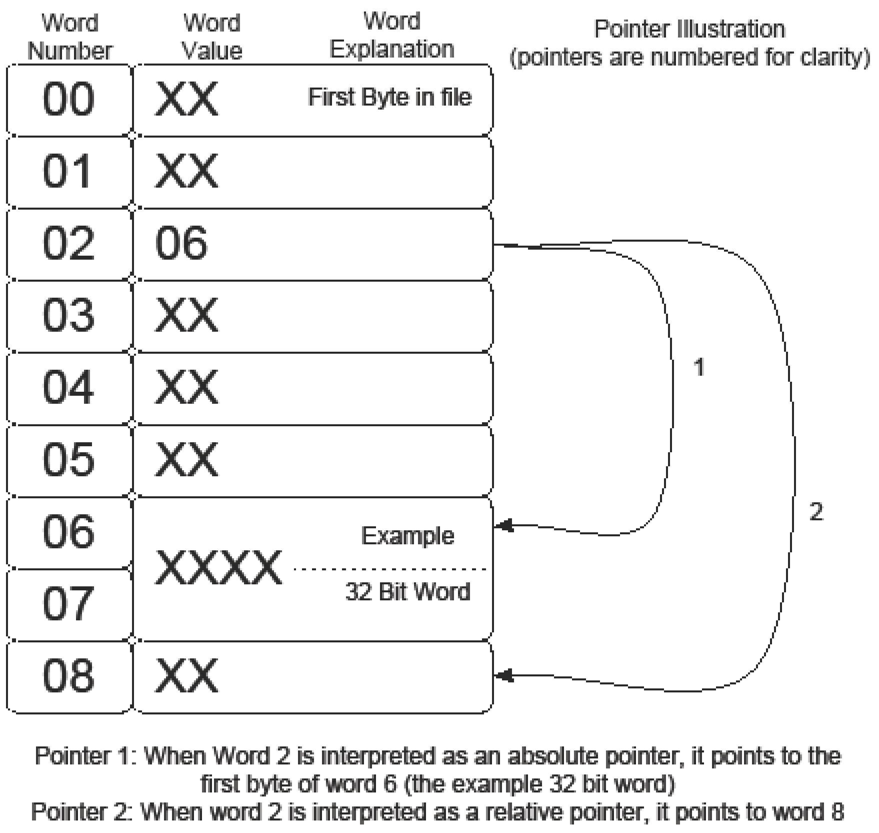

Word Number
Word Value
Word Explanation
Pointer Illustration (pointers are numbered for clarity)
00
XX
First Byte in file
01
XX
02
06
03
XX
04
XX
05
XX
06
XXXX
Example
32 Bit Word
07
08
XX
1
2
Pointer 1: When Word 2 is interpreted as an absolute pointer, it points to the first byte of word 6 (the example 32 bit word)
Pointer 2: When word 2 is interpreted as a relative pointer, it points to word 8

Figure 1.4.4 – Absolute and Relative Pointer Examples

1.5 Target Hardware ID Definition

The Target Hardware (HW) ID definition is based on two hardware classes:

ARINC Specification 429 class: The Target HW ID is the Equipment Code defined in ARINC Specification 429, and represented as four hexadecimal characters with ASCII “0” padding on the left.

Manufacturer’s class: The specific Target HW ID may use 4 to 15 characters with the first three characters reflecting the Manufacturer’s Code. The manufacturer should administer the remaining characters.

COMMENTARY

Target HW compliant with ARINC 615 should be identified by using the ARINC 429 equipment identification code. Target HW compliant with ARINC 615A should be identified using the Manufacturer’s Code (MMM) identification.

Target HW ID is used by loaders to link parts to selected load destinations and vice-versa. The target hardware may use Target HW ID to ensure that incoming loads are compatible.

Increased use of Commercial Off The Shelf (COTS) software and Integrated Modular Avionics (IMA) makes it desirable to have software parts that may be applied across multiple LRUs. In this case, a generic Target HW ID should be chosen for the software part to prevent software part number changes when new hardware can accept existing software parts.

Suppliers are encouraged not to specify multiple Target HW IDs for an LRU, since this increases required Load activity. Systems with multiple internal channels should manage redundant loads to all channels internally and not require mechanics to perform multiple loads for each LSP.

1.6 Electronic Distribution

Section 2 of this document defines the structure of distinct LSPs and related Batch Parts. These parts are independent of any form of transport media or delivery method. As such, this definition serves as the basis for Electronic Distribution of Software (EDS) parts.

Section 3 of this document defines the structure for assembling LSPs and associated batch files onto physical transport media to create loadable software Media Set Parts. These specifications facilitate controlled distribution of Media Set Parts, thus serving to ensure secure delivery of LSPs and Batch Parts contained on that media.

Under EDS, LSPs and Batch Parts may be delivered irrespective of specifications in Section 3, where the EDS method provides equivalent security as that offered in Section 3, ensuring parts received are exactly that which were sent.

Media Sets, recognized as Parts themselves may also be delivered by way of EDS. In this scenario the physical media may be re-created as exact duplicates of that sent, retaining the integrity of the LSPs contained on the media.

2.0 LOADABLE SOFTWARE PARTS

2.1 Software Load PN

Each LSP should have only one Part Number (PN). Both the aircraft manufacturer and the supplier of the software should mutually agree upon the PN.

A new unique PN should be assigned to the part any time a change is made to an LSP.

COMMENTARY

Any bit change in the LSP (even if the data is not actually transferred into the unit at load time) requires that a new PN be assigned to the load. If the same software PN has been assigned to two software parts with two different bit images, then there is a risk that the wrong bit image might find its way into an inappropriate situation.

2.1.1 Software Load PN Format

The format for Loadable Software PNs should be MMMCC-SSSS-SSSS, where:

MMM is a unique, upper-case alphanumeric identifier that is assigned to each software supplier. See Section 2.1.2

CC is two “check characters” generated from the other characters in the PN, as defined in Section 2.1.3.

SSSS-SSSS is a software supplier defined unique product identifier consisting of upper-case alphanumeric characters except for alpha characters “I,” “O,” “Q,” and “Z.” The Load PN should have no embedded blanks.

“-” Hyphens (ASCII 0x2D) are delimiters and are included as part of the software PN as indicated above. Delimiters do not contribute to the uniqueness of the number.

ARINC 615A loaders should not implement checks for compliance with the specific PN format rules. ARINC 615A loaders should be able to process loads that are not fully compliant with the PN characteristics defined herein (e.g., existence/placement of delimiters, characters used and other format variations). This enables maximum backward compatibility and flexibility without creating future compatibility problems.

COMMENTARY

Approximately one trillion (1,000,000,000,000) PNs are available for each supplier identification code (MMM) to be managed by the supplier’s configuration control organization. The intent is not to allocate new identification codes to suppliers for new programs, rather a supplier is expected to continue to use the allocated MMM code until all numbers are used up. Thus, suppliers should not allocate large blocks of PNs when only a few are needed.

The MMMCC-SSSS-SSSS format may create numbers that technically do not meet two Air Transport Association (ATA) part number format requirements. This report acknowledges the potential conflicts with ATA 2000, because of more important considerations. Comments concerning potential conflict areas follow:

- ATA 2000 specifies that delimiters should not be placed next to letters.
- The bulk of pre-existing LSPs are not constrained by this ATA delimiter/character restriction. However, limited LSP

management systems have been found to impose this rule. Under these circumstances, suppliers should select SSSS-SSSS values that preclude letters from occurring in positions 7, 10, and 12 of the PN, and which result in a number value in the second position of the Load PN CC field.

- ATA 2000 specifies that the letter “O” should not be used.
- Some MMM codes contain a letter “O” (e.g., COL is assigned to the Air Transport Division of Rockwell Collins, Inc.), which have not produced errors. For MMM codes, commonly confused characters should be recognized as Alphabetic, by default, i.e., “O” rather than zero, “I” rather than one.

2.1.2 Manufacturer’s Codes Assignment

The Manufacturer’s Code (MMM) is an identification code assigned to each organization that develops aircraft software. Three upper-case alphanumeric characters comprise the code. Attachment 1 describes how an LSP provider can apply for a Manufacturer’s Code. The list of MMM Codes is posted on the ARINC Website with a link from the AEEC Webpage at the URL:

http://www.aviation-ia.com/aeec/projects/manufacturer_code/index.html

ARINC Industry Activities serves as the administrator of MMM codes.

COMMENTARY

The software part numbering system is intended for decades of use. Thus, the assignment of MMM codes conserves PNs within each MMM code block.

To avoid the proliferation of MMM codes, only one MMM code is assigned to each organization.

In some cases, a given organization may be composed of more than one subsidiary that has an independent configuration control organization. In this case, it is acceptable to assign more than one MMM code to each organization. Otherwise, multiple MMM codes may be assigned only when an organization can show that they have depleted the number of PNs for their designator.

Airframe manufacturers are expected to monitor the use of MMM codes and work together to resolve any PN assignment problems that might arise, e.g., LSP numbers with MMM codes that are not formally assigned to the creator of the software.

The role of the MMM administrator is to:

- Maintain a database of all assigned MMM codes, which is posted on the ARINC Industry Activities website.
- Assign MMM codes upon written request, to organizations according to the guidelines and restrictions provided in this document.
- Publish the list of the approved MMM codes.

Written requests should be submitted to the ARINC 665 Manufacturer's Code Administrator. Appendix J provides ~~a form~~**the methods** for requesting a Manufacturer's Code assignment.

2.1.3 Check Characters in the Software PN

~~The purpose of the Check Characters (CC) is to increase the integrity of the aircraft configuration report. The FAA/JAA concern being addressed by the CCs is that an incorrect PN reported by a system might be corrupted by a lower integrity display system in a manner that causes it to be displayed as correct. See Appendix E for computation methods.~~

Check Characters (CC) should be computed as specified in Section 4.3.1 and Appendix E.

2.1.4 Commercial Software

The use of commercial software not specifically designed for airplane applications may be incorporated as an integral part of airplane systems. Suppliers of systems incorporating commercial software will do so under the identification of a Software Load PN compliant with Section 2.1.1. The system supplier will include their own MMM code in that PN. Patch upgrades to commercial software necessitates release of a new PN.

2.2 Software Load Content and Format

2.2.1 Software Load Structure

A load consists of a Header File plus one or more Data Files. A load may also include support files as needed. See Appendix A, Figure A.1 for a load structure diagram. File names within a load should be unique. File names within ARINC 665 header files should be treated as if they were case-sensitive.

The Header File and each Data File should consist of an integral number of 16-bit words. It is recommended (but not required) that all Support Files also consist of an integral number of 16-bit words. As a minimum, Support Files should consist of an integral number of 8-bit bytes.

COMMENTARY

Appendix F, Implementation for Multi-Standard Compatibility, contains specific part format, filename, and media set requirements when creating loads that can be loaded by ARINC 615A loaders, ARINC 615-2 and later loaders, and Boeing 777 ARINC 629 loaders.

It is important to note the independence of an LSP from a Media Set Part (MSP). LSPs may be separated from MSPs for electronic storage and distribution.

LSPs should be created without including any data that ties the LSP to any MSP structure, loadable media set parts, or directory structure external to the LSP.

LSPs can be managed independent of any form of loadable media set parts or delivery method. The independence of LSPs from MSP structure allows for electronic distribution of LSPs.

LSP header, data, and support files cannot be dependent on placement within directory structures on support tools since some transport protocols (e.g., some e-mail, ARINC 615A) cannot transport directories. Also, support tools may need to restructure the organization of the LSP files for transport or

storage as needed. Note that the LSP header file does not allow path name references.

However, LSP data and support files can contain internal directory structure if the directory structure is packaged into a single data or support file (e.g., ZIP format). This allows the LSP data and support files to be easily handled on the ground and loaded to target hardware (within which the directories can be expanded).

For more information, see ARINC Project Paper 641: *Logical Software Part Packaging for Transport*.

LSPs should not contain file names that only have a difference in uppercase and lower-case letters, e.g., “file” and “File.”

2.2.2 Software Load File Naming

The File name for the Header, Data, and Support files that comprises the load should be a maximum of 255 characters long including delimiter “.” and extension characters.

The first three characters of the Header filename should be the Manufacturer’s Code of the creator file, described in Sections 2.1.1 and 2.1.2. The rest of the filename should be assigned such that it is unique for each load associated to the Manufacturer’s Code.

~~Restriction of characters in the header, data and support file, and directory names is defined to avoid non-printable characters in filenames, or filenames, which are not cross-platform usable. Data and support filenames should only consist of printable characters, excluding “~”, “/”, “:”, “\”, “I”, “O”, “Q”, and Blank. The filenames “.” And “..” are not allowed.~~

Restriction of characters in the header, data, and support file names is defined to avoid non-printable characters in filenames, or filenames which are not cross-platform usable.

Filenames shall not contain the English letters “I (i), O (o), or Q (q).

Filenames shall not contain the following restricted subset of 7-bit US-ASCII printable characters:

Space	*	\	/	“	<	>	?	|	~

The filenames “.” and “..” are not allowed.

Data files and support files should have unique file names across all LSPs that may be loaded to a particular target hardware if the target hardware is loaded via ARINC 615A data loading protocol during one uninterrupted load session.

File references in ARINC 665 formatted files should match the case of the referenced files.

COMMENTARY

The properties of the underlying file system of an actual media set may impose additional constraints for file naming.

A chosen file name may be valid for the file system of one media type and invalid for another. An example would be the file name

"123456789.LUP". This file name is valid for an ISO file system of a data CD, but not for a DOS 3.1 floppy file system.

File names on ARINC 665 media and references to them from within ARINC 665 files are case-sensitive. The properties of the underlying file system of an actual media set may not be able to handle case sensitivity. For example, the two file names "ABC" and "abc" are completely distinct. Two files of these names may reside within the same directory in the case that the underlying file system supports case sensitivity.

2.2.2.1 Header File Name Extension

The LSP Header Filename Extension should be “LUH.”

2.2.2.2 Data File Name Extension

In general, Data File Name extensions are user defined and can technically be anything that does not violate the Section 3.2.2 list of reserved extensions. It is highly recommended that Data File Name extensions be “LUP.”

2.2.2.3 Support File Name Extensions

Support Files extensions are user defined and can be named anything that does not violate the list of reserved extensions included in Section 3.2.2.

2.2.3 File Content and Format

2.2.3.1 Header File Content and Format

The Header File for each LSP should contain the information defined in Table 2.2.3-1, Header File Content.

The placement of the fields**,** respecting byte significance and conditional use of NUL values within the Load Header File*,* should be as defined in Figure C-1 of Appendix C, Header File Format.

All values should be expressed as binary numbers except the noted ASCII character fields.

Detailed Field descriptions are listed and explained in the order they appear in Table 2.2.3-1.

Table 2.2.3-1 – Header File Content

Name of Field	Field Size (bits)	Note
Header File Length	32	
Load File Format Version	16	
Part Flags	16	
Pointer to Load PN Length	32	
Pointer to Number of Target HW IDs	32	
Pointer to Number of Data Files	32	
Pointer to Number of Support Files	32	
Pointer to User Defined Data	32	
Pointer to Load Type Description Length	32	
Pointer to Number of Target HW ID with Positions	32	
Pointer to Load Check Value Length	32	
Expansion Point No.1	0	
Load PN Length	16	
Load PN	16	1
Expansion Point No. 2	0	
Load Type Description Length	16	2
Load Type Description	16	1, 2
Load Type ID	16	2
Expansion Point No. 3	0	
Number of Target HW IDs	16	
* Target HW ID Length	16	
* Target HW ID	16	1
Expansion Point No. 4	0	
Number of Target HW ID with Positions	16	2
% Target HW ID with Positions Length	16	2
% Target HW ID with Positions	16	1, 2
% Number of Target HW ID Positions	16	2
%& Position Length	16	2
%& Position	16	1, 2
Expansion Point No. 5	0	
Number of Data Files	16	
+ Data File Pointer	16	
+ Data File Name Length	16	
+ Data File Name	16	1
+ Data File PN Length	16	
+ Data File PN	16	1
+ Data File Length	32	
+ Data File CRC	16	
+ Data File Length in Bytes	64	
+ Data File Check Value Length	16	
+ Data File Check Value Type	16	2
+ Data File Check Value	16	1, 2
+ Expansion Point No. 6	0	
Expansion Point No. [ ]	**0**	
Number of Support Files	16	2
# Support File Pointer	16	2
# Support File Name Length	16	2
# Support File Name	16	1, 2

Name of Field	Field Size (bits)	Note
# Support File PN Length	16	2
# Support File PN	16	1, 2
# Support File Length	32	2
# Support File CRC	16	2
# Support File Check Value Length	16	2
# Support File Check Value Type	16	2
# Support File Check Value	16	1, 2
# Expansion Point No. 7	0	
Expansion Point No. 8	0	
User Defined Data	Multiples of 16	1, 2
Expansion Point No. 9	0	
Load Check Value Length	16	
Load Check Value Type	16	2
Load Check Value	16	1, 2
Header File CRC	16	
Load CRC	32	

Notes:

1. ~~One or more 16-bit words.~~**Variable length field of 16-bit words.**
2. ~~Zero or more 16-bit words.~~**Optional and is omitted if the associated pointer field is zero (see field descriptions).**

* Fields repeated as a group for each Target HW ID.

% Fields repeated as a group for each Target HW ID with Positions.

& Fields repeated as a group for each Position within a Target HW ID with Positions group.

+ Fields are repeated as a group for each Data File.

# Fields are repeated as a group for each Support File. If no support files are included in the load, then these fields are omitted.

2.2.3.1.1 Header File Length

Header File Length is defined as the number of 16-bit words in the header file including this field.

2.2.3.1.2 Load File Format Version

The Load File Format Version is defined by a 16-bit word as directed in Section 1.4.1, File Format Version Definition.

2.2.3.1.3 Part Flags

The Part Flags are defined by a 16 bit word. They are used to indicate extra information to help operators and systems distinguish and understand the purpose of a part.

2.2.3.1.3.1 Download Flag

The Least Significant Bit is used to indicate an upload/download part. The value of 0 indicates an upload. The value of 1 indicates that the part contains instructions for download. A dataloader may use this field to help the operator select parts that correspond with a~~n~~ operation.

COMMENTARY

In an upload operation, a dataloader uses information from the LSP to cause a transfer of specific information from the LSP to the target. In a download operation, a dataloader uses information from the LSP to cause a transfer of specific information from the target to the dataloader.

The purpose of assigning a PN to a download LSP is to identify, control, and provide integrity for the files in the LSP which describe to the dataloader how to perform the specific download operation. The PN of the download LSP only pertains to the files that are provided to the dataloader which cause the download. The download PN does not provide configuration data for the data downloaded from the target.

2.2.3.1.3.2 Spare Flags

The other 15 bits of the 16 bit Part Flags field are reserved for future use and should be set to binary 0.

2.2.3.1.4 Pointer to Load PN Length

This is an absolute pointer (number of 16-bit words from start of file) to the Load PN Length field.

2.2.3.1.5 Pointer to Number of Target

This is an absolute pointer (number of 16-bit words from start of file) to the Number of Target HW IDs field.

2.2.3.1.6 Pointer to Number of Data Files

This is an absolute pointer (number of 16-bit words from start of file) to the Number of Data Files field.

2.2.3.1.7 Pointer to Number of Support Files

This is an absolute pointer (number of 16-bit words from start of file) to the Number of Support Files field.

Set the value to 0x0000 if there are no support files and omit the Number of Support Files field as well as subordinate Support Files fields; those listed with a prefix of “#” in the table of Header File Content (Table 2.2.3-1).

2.2.3.1.8 Pointer to User Defined Data

This is an absolute pointer (number of 16-bit words from start of file) to the first word of the User Defined Data field. Set the value to 0x0000 if there is no user defined data field.

2.2.3.1.9 Pointer to Load Type Description Length

This is an absolute pointer (number of 16-bit words from start of file) to the first word of the Load Type Description Length field. Set the value to 0x0000 if there is no Load Type Description field.

2.2.3.1.10 Pointer to Number of Target HW ID with Positions

This is an absolute pointer (number of 16-bit words from start of file) to the first word of the Number of Target HW ID with Positions field. Set the value to 0x0000 if there is no Target HW ID with Positions field.

This field should be set to zero if the Number of Target HW ID with Positions field is not used. In this case, the Number of Target HW ID with Positions field, and subordinate target hardware with positions field identified with a prefix of"%" and "%&" in Table 2.2.3-1, will be omitted.

2.2.3.1.11 Pointer to Load Check Value Length

This is an absolute pointer to the Load Check Value Length field.

2.2.3.1.12 Expansion Point No. 1

This is a point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.13 Load PN Length

This is the number of 8-bit ASCII characters in the load PN, including delimiters. This number does not include any NULs appended to fill out the field if the number of characters in the Load PN is odd.

2.2.3.1.14 Load PN

This field contains the string of 8-bit ASCII characters representing the Load PN whose length is defined by the Load PN Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

Implementers should ensure that the PN is compliant with the recommendations of Section 2.1.1, Software Load PN Format.

2.2.3.1.15 Expansion Point No. 2

This is a point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.16 Load Type Description Length

This is the number of 8-bit ASCII characters in the Load Type Description. This number does not include any NULs appended to fill out the field if the number of characters in the Load Type Description is odd. This field is omitted if the Pointer to Load Type Description Length is set to 0x0000.

2.2.3.1.17 Load Type Description

This field contains the string of 8-bit ASCII characters representing the Load Type Description, whose length is defined by the Load Type Description Length field. This field is omitted if the Pointer to Load Type Description is set to 0x0000.

The field is allocated on an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

The Load Type Description string describes the load or the function the load performs (e.g., "EEC Operational Software," "FMS Navigation Data Base," etc.).

2.2.3.1.18 Load Type ID

The Load Type ID is a 16-bit hexadecimal numeric value set by the manufacturer or system integrator. The value in this field should correspond with the content of the Load Type Description field. This field is omitted if the Pointer to Load Type Description is set to 0x0000.

Load Type ID should be unique for each LSP type which is loaded to a particular target hardware type.

COMMENTARY

Load Type ID is used to easily identify the software part type. This allows the target to identify which load the incoming load replaces and where to place it in memory.

Operational Software Loads could have a value of 0x0001, FMS Navigation Data Base Loads could have a value of 0x0020, etc. The specific Load Type Description strings and Load Type ID values are left up to the product manufacturer or system integrator for a particular program. All of the loads that have different Load Type Descriptions for a particular program would have different Load Type ID values. All the loads that have the same Load Type Description would have the same Load Type ID value.

Load Type Descriptions and Load Type IDs could vary from program to program; however, loads with similar Load Descriptions, yet on different programs, could have the same Load Type ID value. That is, “EEC Operational Software” could have a Load Type ID value of 0x0001, and “FMC Operational Software” could also have a Load Type ID of 0x0001, where both loads are operational software loads. This only works as long as “EEC Operational Software” and “FMC Operational Software” are never loaded in the same Target HW.

2.2.3.1.19 Expansion Point No. 3

This is a point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.20 Number of Target HW IDs

This is the number of Target HW IDs in the following Target HW ID list. Refer to Section 1.5 for use of Target HW IDs.

2.2.3.1.21 Target HW ID Length

This is the number of characters in the Target HW ID. This number does not include any NULs appended to fill out the field if the number of characters in the Target HW ID is odd.

2.2.3.1.22 Target HW ID

This field contains the string of 8-bit ASCII characters representing a Target HW ID whose length is defined by the Target HW ID Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

2.2.3.1.23 Expansion Point No. 4

This is a point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.24 Number of Target HW ID with Positions

This is the Number of Target HW IDs with Positions in the following Target HW IDs with Positions list. If the LSP is applicable to all positions of Target HW IDs listed in the Target HW ID list, then this, and subsequent related fields should be omitted and the Pointer to Number of Target HW ID with Positions is set to 0x0000.

COMMENTARY

Target HW ID with Position is not intended to replace the Target HW ID defined above, which remains mandatory.

Target HW ID with Positions is only used to restrict the LSP upload into a specific position of a Target HW ID (e.g., allow upload only into equipment in the left position but not in the right position).

2.2.3.1.25 Target HW ID with Positions Length

This is the Number of 8-bit ASCII characters in the Target HW ID with Positions field. This number does not include any NULs appended to fill out the field if the number of characters in the Target HW ID is odd. This field is omitted if the Pointer to Number of Target HW ID with Positions is set to 0x0000.

2.2.3.1.26 Target HW ID with Positions

This field contains the string of 8-bit ASCII characters representing the Target HW ID with Positions whose length is defined by the Target HW ID with Positions Length field. This field is omitted if the Pointer to Number of Target HW ID with Positions is set to 0x0000.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string. This field follows the same definition rules as the Target HW ID.

This field should also match one of the Target HW IDs listed in the Target HW ID list.

2.2.3.1.27 Number of Target HW ID Positions

This is the number of Target HW ID Positions in the following Position list. If there are no Target HW IDs with Positions, then this field should be omitted if the Pointer to Number of Target HW ID with Positions is set to 0x0000.

2.2.3.1.28 Position Length

This is the number of characters in the target hardware ***Position*** field. This number does not include any NULs appended to fill out the field if the number of characters in the Position field is odd. This field is omitted if the Pointer to Number of Target HW ID with Positions is set to 0x0000.

2.2.3.1.29 Position

This field contains the string of 8-bit ASCII characters representing a target hardware Position, for which the LSP is intended, whose length is defined by the Position Length field. This field is omitted if the Pointer to Number of Target HW ID with Positions is set to 0x0000.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

COMMENTARY

The Position field identifies the instantiation of the Target HW ID in the system. The Position field can be “L,” “R,” “1,” “02,” “3F,” “5-C,” etc., as defined by the system integrator.

2.2.3.1.30 Expansion Point No. 5

This is the point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.31 Number of Data Files

This is the number of Data Files in the software load. The value must be greater than zero since there must be at least one data file in each load.

It is not necessary for Data Files to be listed in contiguous, alphanumeric, or any specific order.

2.2.3.1.32 Data File Pointer

This is the relative number of 16-bit words to the next Data File Pointer.

The value of the “Data File Pointer” for the last data file in the list should be 0x0000.

2.2.3.1.33 Data File Name Length

This is the number of characters in the Data File Name. This number does not include any NULs appended to fill out the field if the number of characters in the Data File Name is odd.

2.2.3.1.34 Data File Name

This field contains the string of 8-bit ASCII characters representing the Data file Name whose length is defined by the Data File Name Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

COMMENTARY

Implementers are strongly encouraged to use unique Data File Names within a given load PN.

2.2.3.1.35 Data File PN Length

This is the number of characters in the Data File PN. This number does not include any NULs appended to fill out the field if the number of characters in the Data File PN is odd.

2.2.3.1.36 Data File PN

This field contains the string of 8-bit ASCII characters representing the Data File PN whose length is defined by the Data File PN Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

~~Implementers should ensure that the Data File PN is unique within a given load PN.~~**It is recommended that LSP suppliers ensure that the data file part number is unique within a given LSP.**

COMMENTARY

The data file part number is assigner by the LSP supplier. The data file part number is intended to be used to manage changes in file content as LSPs are changed over time. Data file part numbers are not used for any LSP configuration control outside of the LSP supplier.

Target hardware can also use data file part numbers of an LSP during the load process to determine whether a file presented for load has changed from the files that are part of its currently installed LSP. This can be used by target hardware to perform a short load as described in ARINC 615A data loading protocol.

2.2.3.1.37 Data File Length

The data file length is the number of 16-bit words in the data file. A half-word at the end of a data file should be counted as a complete word.

2.2.3.1.38 Data File CRC

The Data File CRC is a 16-bit CRC covering the entire Data File. The CRC should be computed as defined in Section 4.

2.2.3.1.39 Data File Length in Bytes

This is the number of 8-bit bytes in the Data File.

COMMENTARY

This value should be used in conjunction with the Data File Length in 16-bit words.

2.2.3.1.40 Data File Check Value Length

This is the number of 8-bit bytes in the Data File Check Value**,** including this field and Data File Check Value Type. The Check Value Length should be implemented as defined in Section 5. Set value to 0x0000 if not used.

2.2.3.1.41 Data File Check Value Type

This indicates the type of Check Value stored in the Data File Check Value. The Check Value Type should be implemented as defined in Section 5. Omit if Data File Check Value Length is set to 0x0000.

2.2.3.1.42 Data File Check Value

This is a variable length and variable data type field containing the Data File Check Value. The Check Value should be implemented as defined in Section 5, consistent with the Date File Check Value type. Omit if Data File Check Value Length is set to 0x0000.

2.2.3.1.43 Expansion Point No. 6

This is a point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.44 Number of Support Files

This is the number of Support Files in the software load.

If the Pointer to the Number of Support Files field is set to 0x0000, then the Number of Support Files field and subordinate Support File fields should be omitted from the file.

2.2.3.1.45 Support File Pointer

The Support File Pointer is the relative number of 16-bit words to the next Support File Pointer.

The value of the "Support File Pointer" for the last support file in the list should be 0x0000.

2.2.3.1.46 Support File Name Length

This is the number of characters in the Support File Name. This number does not include any NULs appended to fill out the field if the number of characters in the Support File Name is odd.

2.2.3.1.47 Support File Name

This field contains the string of 8-bit ASCII characters representing the Support File Name whose length is defined by the Support File Name Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

COMMENTARY

Implementers are strongly encouraged to use unique Support File Names within a given load PN***.***

2.2.3.1.48 Support File PN Length

This is the number of characters in the Support File PN. This number does not include any NULs appended to fill out the field if the number of characters in the Support File PN is odd.

If no part number is assigned to a support file, this field should be set to zero.

2.2.3.1.49 Support File PN

This field contains the string of 8-bit ASCII characters representing the Support File PN whose length is defined by the Support File PN Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

If the Support file PN Length is 0x0000, then this field should be omitted from the file. ~~The PN should be compliant with Section 2.1.1, Software Load PN Format.~~**It is recommended that LSP suppliers ensure that the support file part number is unique within a given LSP.**

Implementers should ensure that the support file PN is unique within a given load PN.

2.2.3.1.50 Support File Length

This is the number of 8-bit bytes in the Support File.

2.2.3.1.51 Support File CRC

The Support File CRC is a 16-bit CRC covering the entire Support File. The CRC should be computed as defined in Section 4.

2.2.3.1.52 Support File Check Value Length

This is the number of 8-bit bytes in the Support File Check Value including this field and Support File Check Value Type. The Check Value Length should be implemented as defined in Section 5. Set value to 0x0000 if not used.

2.2.3.1.53 Support File Check Value Type

This indicates the type of Check stored in the Support File Check Value. The Check Value Type should be implemented as defined in Section 5. Omit if support File Check Value Length is set to 0x0000.

2.2.3.1.54 Support File Check Value

This is a variable length and variable data type field containing the Support File Check Value. The Check Value should be implemented as defined in Section 5, consistent with the Support File Check Value type. Omit if Support File Check Value Length is set to 0x0000.

2.2.3.1.55 Expansion Point No. 7

This is a point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.56 Expansion Point No. 8

This is a point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.57 User Defined Data

This is the user defined area containing data defined at the discretion of the LSP supplier. This field may be omitted.

2.2.3.1.58 Expansion Point No. 9

This is a point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.2.3.1.59 Load Check Value Length

This is the number of 8-bit bytes in the Load Check Value including this field and Load Check Value Type. The Check Value Length should be implemented as defined in Section 5. Set value to 0x0000 if not used.

2.2.3.1.60 Load Check Value Type

This indicates the type of Check Value stored in the Load Check Value field. The Check Value Type should be implemented as defined in Section 5. Omit if Header File Check Value Length is set to 0x0000.

2.2.3.1.61 Load Check Value

This is a variable length and variable data type field containing the Load Check Value. The Load Check Value should be implemented as defined in Section 5, consistent with the ~~Support File~~ Load Check Value type. ~~Omit if Header File Check Value Length is set to 0x0000~~**The Load Check Value should be omitted if Load Check Value Length is set to zero.**

The Load Check Value covers the entire Software Load, including all Data Files, Support Files and Header File contents excluding the Load Check length Load Check Value type, the Load Check Value fields, the Header File CRC and the Load CRC.

2.2.3.1.62 Header File CRC

The Header File CRC is a 16-bit CRC covering fields in the Header file, excluding the Header File CRC and the Load CRC field.

The Header File CRC should be computed as defined in Section 4.

2.2.3.1.63 Load CRC

The Load CRC is a 32-bit CRC covering the entire Software Load, including all Data Files, Support Files and Header File contents excluding the "Load CRC" itself.

The Load CRC should be computed and placed in the header file after the Header File CRC is calculated and inserted into the Header File.

The Load CRC should be computed as defined in Section 4**, and calculated in the following order:**

1. **Header file contents excluding the Load CRC field**
2. **Data files in the sequence they are listed in the header file**
3. **Support files in the sequence they are listed in the header file**

2.2.3.2 Data File Content and Format

The content of a data file is entirely up to the supplier of the software load. The format of the data file content is also up to the supplier of the software load, with the single exception that each data file should contain an integral number of **bytes. The format of the data file is chosen by the LSP supplier.**~~16-bit words.~~

2.2.3.3 Support File Content and Format

The content of any Support File is entirely up to the creator of the software load. The format of the support file content is also up to the creator of the software load, with the single exception that each support file should contain an integral number of 8-bit words.

COMMENTARY

Typically, the CONFIG.LDR file, defined in ARINC Report 615-3, should be used as an ARINC 615A support file to obtain media compatibility.

2.2.4 Data and Support File Options

Within the User Defined Data field of the Header File, additional information may be included to manage the data transfer operation.

2.2.4.1 File Compression

Data or Support Files may optionally be compressed in order to save media space, and to save transmission time when being loaded. The Header File should not be compressed since loaders and other tools require access to this information.

If data compression is used, the implementer should consider embedding a CRC of the uncompressed File in the File before compressing it.

COMMENTARY

Implementers should consider embedding a CRC of the software load with uncompressed data files in the User Defined Data field of the Header File.

The purpose of these CRCs is to enable the target HW to determine the validity of the software load (and Files) after decompression.

The target HW would convert the file to its original, expanded form before storing in program memory and verifying software load/file validity, etc.

All uses of Load, Data, or Support File CRC, and Check Value should be computed using the final form of the Data or Support File (after compression).

2.2.4.2 File Encryption

Data or Support Files may optionally be encrypted. The Header File should not be encrypted since loaders and other tools require access to this information.

If data encryption is used, the implementer should consider embedding a CRC of the unencrypted File in the File before encrypting it.

COMMENTARY

The purpose of these CRCs is to enable the target HW to determine the validity of the software load (and Files) after de-encryption.

Implementers should consider embedding a CRC of the software load with non-encrypted files in the User Defined Data field of the Header file.

The target would convert the file to its original, non-encrypted form before storing in program memory and verifying load/file validity.

All use of Load, Data or Support File CRC, and Check Values in the HEADER and FILES.LUM files should be computed using the final form of the Data and and/or Support File (after encryption).

2.3 Optional Files

This section defines the optional file formats used by the ARINC 615A data loading system.

2.3.1 Batch File

There is a desire by the airlines to be able to define a “batch” type file that enables the maintenance person to select a file that defines for the Data Loader a series of loads that should be loaded into one or more Target HW. This Batch File should enable the maintenance person to not have to select all the loads that are desired to be loaded into each of those Target HW positions.

The Batch File is based on the Load-List block notion, which is that one Load-List block defines all the loads that belong to one Target HW ID position. More than one Load-List block can be included in the Batch File.

The Batch File should be identified as: <Batch File>.LUB.

Batch File is identified in the FILES.LUM file.

The placement of Batch File Parts (BFP) fields, respecting byte significance and conditional use of NUL values within the BFP, should be as defined in Figure C-6 of Appendix C, Batch File Part Format.

Header file names referenced within BFP files should match the case of the actual file names.

The first three characters of the BFP name should be the Manufacturer's Code of the creator file as described in Sections 2.1.1 and 2.1.2. The remainder of the BFP name should be assigned as such that is unique for each BFP defined by the Manufacturer's Code. Refer to Section 2.2.2 for allowable ASCII characters in file names.

Header file names referenced within a BFP should be the complete name of the header files, including extensions and without any path reference.

Batch files are distributed to onboard and off-board ARINC 615A software data loaders and are used by the loaders only to automate the setup of multiple LSP loads. BFPs are not transferred to target hardware.

COMMENTARY

Since the Batch File is meant to replace the selections of destinations and source that a maintenance person would have to make on a Data Load, the position should be included with the Target HW ID (THW_ID_POS as defined in ARINC 615A).

The Batch File should contain the information defined in Table 2.3.1-1.

Table 2.3.1-1 – Batch File Content

Name of Field	Field Size (bits)	Note
Batch File Length	32	
Batch File Format Version	16	
Spare	16	
Pointer to Batch File PN Length	32	
Pointer to Number of Target HW ID Load-list Blocks	32	
Expansion Point 1	0	
Batch File PN Length	16	
Batch File PN	16	1
Comment Length	16	
Comment	16	1, **2**
Expansion Point 2	0	
Number of Target HW ID Load-List Blocks	16	
+ Pointer to next Target HW ID Load-List Block	16	
+ Target HW ID POS Length	16	
+ Target HW ID POS	16	1
+ Number of Loads for this Target HW ID POS	16	
+ Header File Name Length	16	
+ Header File Name	16	1
+ Load PN Length	16	
+ Load PN	16	1
Expansion Point 3	**0**	
Batch File CRC	16	

Note:

1. ~~One or more 16-bit words~~**Variable filed length of 16-bit words.**

2. **Optional field, omitted if the associated length field is set to zero (see field descriptions).**

+ Field is repeated for each Target HW ID Load-List Block.

+ # Field is repeated for each load of one Target HW ID.

Detailed field descriptions are listed in the following sections in the order they appear in Table 2.3.1-1.

2.3.1.1 Batch File Length

The Batch File Length is defined as the number of 16-bit words in the batch file.

2.3.1.2 Batch File Format Version

Sixteen bits define the Batch File Format Version. The Batch File Format Version is defined in Section 1.4.1, File Format Version Definition.

2.3.1.3 Spare

The spare field is used to align the pointers that follow, which are defined on 4-byte boundaries.

2.3.1.4 Pointer to Batch File PN Length

This is the absolute pointer (number of 16-bit words from start of file) to the Batch File PN length field.

2.3.1.5 Pointer to Number of Target HW ID Load List Blocks

This is the absolute pointer (number of 16-bit words from start of file) to the Number of Target HW ID Load List Blocks field.

2.3.1.6 Expansion Point 1

This is the point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

2.3.1.7 Batch File Length

The number of characters in the Batch File PN does not include any NULs appended to fill out the field if the number of characters in the Batch File PN is odd.

2.3.1.8 Batch File PN

This field contains the string of 8-bit ASCII characters representing the Batch File PN field whose length is defined by the Batch File PN Length field. The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string. Implementers should ensure that the PN is compliant with Section 2.1.1, Software Load PN Format.

Batch file part numbers should be unique from any LSP part numbers.

2.3.1.9 Comment Length

The number of characters in the comment field should not include any NULs appended to fill out the field if the number of characters in the Comment field is odd. If no Comment is associated to the Batch, this field should be set to 0x0000.

2.3.1.10 Comment

This field contains the string of 8-bit ASCII characters representing the batch Comment whose length is defined by the Comment Length field. The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string. If the Comment Length Field is set to 0x0000, the Comment Field should be omitted.

COMMENTARY

This field may be used to include the batch definition design information or modification history of the Batch File.

2.3.1.11 Number of Target HW ID Load-List Blocks

Number of Target HW ID Load-Lists blocks included in the Batch File.

2.3.1.12 Pointer to Next Target HW ID Load-List Block

The pointer to the first word in the group of data for the next Load-List Block. This is repeated for each Target HW ID Load-List block. Set to 0 in the last Load-List block.

2.3.1.13 Target HW ID POS Length

Target HW ID POS Length does not include the any NULs appended to fill out the field if the number of characters in the Target HW ID POS is odd.

2.3.1.14 Target HW ID POS

This field contains the string of 8-bit ASCII characters representing the Target HW ID POS field whose length is defined by the Target HW ID POS Length field. The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string. ~~Implementers should ensure that the Target HW ID POS is consistent with the Target HW IDs listed in the Header Files of the loads listed under this Target HW ID POS.~~

The Target HW ID POS value should match the THW_ID_POS value of the target hardware to be loaded by this BFP. THW_ID_POS is defined in Section 5.3.4 of ARINC 615A-3, and Section 6.4 of ARINC 615A-2. The definition is intended for software data loaders to identify the target of an LSP load.

It is recommended that the Target HW ID POS values be consistent with the Target HW ID values in the header files of the ARINC 665 LSPs listed under this Target HW ID POS.

2.3.1.15 Number of Loads for the Target HW ID POS

Number of Loads for the Target HW ID POS defines the number of loads in the Load-List Block for the Target HW ID POS.

2.3.1.16 Header File Name Length

Number of characters in the Header File Name should not include any NULs appended to fill out the field if the number of characters in the Header File Name is odd.

2.3.1.17 Header File Name

The Header File Name defines the actual Header File Name including delimiters.

This field contains the string of 8-bit ASCII characters representing the Header File Name whose length is defined by the Header File Name Length field. The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

The header file name shall match the header file name of the LSP identified by the Load PN field specified in Section 2.3.1.19.

Implementers should ensure that the file name is compliant with the recommendations of Section 2.2.2. The “File Name” is the name of the file, without any information relative to its path. A file name should never begin with a backslash nor contain any backslash. File names**,** should include all extensions and delimiters.

2.3.1.18 Load PN Length

Load PN Length is defined by the number of characters in the Load PN. This number does not include any NULs appended to fill out the field if the number of characters in the Load PN is odd.

2.3.1.19 Load PN

The Load PN is defined by the actual Load PN including delimiters.

This field contains the string of 8-bit ASCII characters representing the Load PN whose length is defined by the Load PN Length field. The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string. Implementers should ensure that the Load PN is compliant with the recommendations of Section 2.1.1.

2.3.1.20 Batch File CRC

The Batch File CRC is defined as the 16-bit CRC covering only the Batch File with the Batch File CRC field excluded. The Batch File CRC should be computed as defined in Section 4.

3.0 LOADABLE MEDIA SET PARTS

3.1 Transport Media Part Number Assignment

Each transport media set should have only one Part Number (PN) which is mutually agreed to by both the aircraft manufacturer and the creator of the software.

Each member of a transport media set is uniquely identifiable by the Media Set PN and the member sequence number.

The Media Set PN should be no longer than 15 characters (including delimiters). The media set PN uniquely identifies a particular configuration of the physical media, label and the software content of the media set.

It is recommended that the Media Set PN comply with the ATA 2000 part number rules. It is recommended that the alphabetic characters “I,” “Q,” and “Z” not be used due to potential reader confusion with other characters. Note: ATA 2000 does not allow the use of the letter “O” in PNs.

The Media Set PN should have no embedded blanks.

The last character of the Media Set PN should not be a hyphen (“-”).

3.2 Transport Media Set Format, Content, and Organization

This section defines the format, content and organization of all types of transport media.

A media set consists of from one to two hundred fifty five media items (members of the set).

Each media set should be comprised of members of the same type (i.e., all 3.5” disks, all PC Cards, etc.).

Media labeling should be as specified in Section 3.3.

COMMENTARY

Appendix F contains specific part format, filename, and media set requirements when creating loads that can be loaded by ARINC 615A loaders, ARINC 615-2 loaders, ARINC 615-3 loaders, and Boeing 777 ARINC 629 loaders.

3.2.1 Transport Media Content and Structure

Each media set member contains a list of the loads on the media set (LOADS.LUM file), a list of all the files on the media set (FILES.LUM file), and a list of all the Batch files on the media set (BATCHES.LUM file). See Appendix B, Figure B-1 for a standard Media Set Structure diagram.

The files that comprise the loads (header, data, and support) need not all be contained on the same media set member. However, a given file should be completely contained on a single media member (i.e., files should not be broken across multiple media members).

The List-of-Loads file named LOADS.LUM should be in the root directory of each member of the media set. See Section 3.2.3.1 for file content and organization.

The List-of-Files file named FILES.LUM should be in the root directory of each member of the media set. See Section 3.2.3.2 for file content and organization.

The List-of-Batches file named BATCHES.LUM should be in the root directory of each member of the media set. See Section 3.2.3.3 for file content and organization.

All files should be contained in the first four directory levels of the media member.

COMMENTARY

The purpose of this restriction is to enable merging of independent media directory structures into a single structure without exceeding the maximum allowable. For example, CD-ROMs are limited to eight levels of directory structure. It has been proposed that all the stable software for a given aircraft (or customer fleet) be combined (by the OEM) onto a single CD-ROM for delivery to the airline. This could not be done if a single supplier were to use eight levels.

All the files on the media set should be listed in the FILES.LUM, except for the FILES.LUM itself (LOADS.LUM should be listed in the FILES.LUM; if present on the media set, BATCHES.LUM should be listed in the FILES.LUM file). The media set may contain files which are not components of any load, which should be listed in the FILES.LUM file.

3.2.2 File Name Extensions

The following file name extensions are reserved for specific file usage and should not be assigned to files other than as defined in Table 3.2.2-1.

Table 3.2.2-1 – File Name Extensions

Ext.	Comment
.CRC	Used for original Boeing standard NON_LOAD.CRC file.
.DIR	Used for original Boeing standard media directory file.
.HDR	Used for original Boeing standard load header file.
.LDR	Used for an ARINC 615 CONFIG.LDR file.
.LCI	Load Configuration Initialization: Defined by ARINC Report 615A.
.LCL	Load Configuration List: Defined by ARINC Report 615A.
.LCS	Load Configuration Status: Defined by ARINC Report 615A.
.LNA	Load Download Answer
.LND	Load Download Disk defined: Defined by ARINC Report 615A.
.LNL	Load Download List: Defined by ARINC Report 615A.
.LNO	Load Download Operator defined: Defined by ARINC Report 615A.
.LNR	Load Download Request: Defined by ARINC Report 615A.
.LNS	Load Download Status: Defined by ARINC Report 615A.
.LUB	Load Upload Batch: Defined by ARINC Report 665
.LUH	Load Upload Header: Defined by ARINC Report 665.
.LUI	Load Upload Initialization: Defined by ARINC Report 615A.
.LUM	Load Upload Media: Defined by ARINC Report 665.
.LUP	Load Upload Part (Data File): Defined by ARINC Report 665.
.LUR	Load Upload Request: Defined by ARINC Report 615A.
.LUS	Load Upload Status: Defined by ARINC Report 615A.

3.2.3 File Content and Organization

3.2.3.1 List of Loads File Content and Organization

The purpose of the LOADS.LUM file is to provide an efficient access to basic information about each load contained on the media set.

The LOADS.LUM file should contain the information defined in Table 3.2.3-1. The placement of the fields within the LOADS.LUM file should be as defined in Appendix C, Figure C-4, LOADS.LUM File Format.

The LOADS.LUM file should list every load on the media set.

Any unused field (e.g., spare field) should be set to a bit image of all zeros.

The LOADS.LUM file on each member of a media set should be identical except for the media sequence number and the LOADS.LUM file CRC fields.

Detailed field descriptions are defined in the following sections in the order they appear in Table 3.2.3.1.

Table 3.2.3.1 - LOADS.LUM File Content

Name of Field	Field Size (bits)	Note
LOADS.LUM File Length	32	
Media File Format Version	16	
Spare	16	
Pointer to Media Set PN Length	32	
Pointer to Number of Loads	32	
Pointer to User Defined Data	32	
Expansion Point No. 1	0	
Media Set PN Length	16	
Media Set PN	16	1
Media Sequence Number (X)	8	
Number Of Media Set Members (Y)	8	
Number of Loads	16	
+ Load Pointer	16	
+ Load PN Length	16	
+ Load PN	16	1
+ Header File Name Length	16	
+ Header File Name	16	
+ Member Sequence Number	16	
+ Number of Target HW IDs	16	
+* Target HW ID Length	16	
+* Target HW ID	16	1
+ Expansion Point No. 2	0	
Expansion Point No. 3	0	
User Defined Data	Multiples of 16	2
LOADS.LUM File CRC	16	

Notes:

1. One or more 16-bit words.
2. Zero or more 16-bit words.

+ Fields are repeated as a group for each load in the media set.

* Fields are repeated as a group for each Target HW ID.

All values should be expressed as binary numbers except the noted ASCII character fields.

3.2.3.1.1 LOADS.LUM File Length

The LOADS.LUM File Length is the number of 16-bit words in the LOADS.LUM file, including this field.

3.2.3.1.2 Media File Format Version

The Media File Format Version is defined by 16-bits. The Media File Format Version is defined in Section 1.4.1, File Format Version Definition.

3.2.3.1.3 Spare

The spare field is used to align the pointers that follow, which are defined on 4-byte boundaries.

3.2.3.1.4 Pointer to Media Set PN Length

This is the absolute pointer (number of 16-bit words from start of file) to the first word of the “Media Set PN Length” field.

3.2.3.1.5 Pointer to Number of Loads

This is the absolute pointer (number of 16-bit words from start of file) to the first word of the “Number of Loads” field.

3.2.3.1.6 Pointer to User Defined Data

This is the absolute pointer (number of 16-bit words from start of file) to the first word of the “User Defined Data” field. Set to if there is no user defined data field.

3.2.3.1.7 Expansion Point No. 1

This is the point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

3.2.3.1.8 Media Set PN Length

This is the number of characters in the Media Set PN.

This number does not include any NULs appended fill out the field if the number of characters in the Media Set PN is odd.

3.2.3.1.9 Media Set PN

The Media Set PN is defined as the actual Media Set PN including delimiters.

This field contains the string of 8-bit ASCII characters representing the Media Set PN whose length is defined by the Media Set PN Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

Implementers should ensure that the Media Set PN is compliant with the recommendations of Section 3.1.

3.2.3.1.10 Media Sequence Number (X)

This is the number of this specific member in the media set. Members are numbered 1 through 255. Zero (0x0000) is not used to number members.

3.2.3.1.11 Number of Media Set Numbers (Y)

This is the number of media members in the media set. For a set consisting of a single member, X = 1 and Y = 1.

3.2.3.1.12 Number of Loads

This is the number of software loads included in the load list. All loads in the media set should be included in the load list.

3.2.3.1.13 Load Pointer

The Load Pointer is the relative number of 16-bit words to the next Load Pointer.

The value of the “Load Pointer” for the last load in the list should be 0x0000.

3.2.3.1.14 Load PN Length

This is the number of characters in the Load PN.

This number does not include any NULs appended to fill out the field if the number of characters in the Load PN is odd.

3.2.3.1.15 Load PN

The Load PN is defined as the actual Load PN including delimiters.

This field contains the string of 8-bit ASCII characters representing the Load PN string whose length is defined by the Load PN Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL (ASCII 0x00) to the character string.

Although the file format allows definition of up to 65535 characters, implementers should not define this character string to be longer than specified elsewhere in this standard (see Section 2.1.1).

3.2.3.1.16 Header File Name Length

This is the number of characters in the Header File Name field.

This number does not include any NULs appended to fill out the field, if the number of characters in the Header File Name is odd.

3.2.3.1.17 Header File Name

This field contains the string of 8-bit ASCII characters representing the Header File Name whose length is defined by the Header File Name Length field. The Header File Name field should be allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

The “Header File Name” is the complete name of the header file, without any information relative to its path. A file name should neither begin with a backslash nor contain any backslash. File names should include all extensions and delimiters. Implementers should ensure that the Header File Name is compliant with the recommendations of Section 2.2.2.

3.2.3.1.18 Member Sequence Number

This is the sequence number of the media member where the header file for this load is located.

3.2.3.1.19 Number of Target HW IDs

This is the number of Target HW IDs in the list.

3.2.3.1.20 Target HW ID Length

This is the number of characters in the Target HW ID.

This number does not include any NULs appended to fill out the field if the number of characters in the Target HW ID is odd.

3.2.3.1.21 Target HW ID

This field contains the string of 8-bit ASCII characters representing the Target HW ID whose length is defined by the Target HW ID Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL ~~(ASCII 0x00)~~ to the character string.

The list of Target HW IDs, for each software load (the fields marked with an asterisk) should be the same list that appears in the header file of the specific load (Ref: Target HW ID List in “Header File Content and Format,” Section 2.2.3.1).

3.2.3.1.22 Expansion Point No. 2

The size of Expansion Point No. 2 should not cause the Load Pointer to overflow.

3.2.3.1.23 Expansion Point No. 3

This is the point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

3.2.3.1.24 User Defined Data

This is the user defined area. The Pointer to User Defined Data field is set to 0x0000, this field should be omitted.

3.2.3.1.25 LOADS.LUM File CRC

The LOADS.LUM File CRC is a 16-bit CRC covering the entire LOADS.LUM file, excluding the LOADS.LUM File CRC field. The CRC should be calculated as defined in Section 4.

3.2.3.2 List of Files File Content and Format

The purpose of the FILES.LUM file is to determine if a specific file is included on the media set and on which member of the media set it is located. The file is be used to determine path of files in the media member. Path file information is supported only by this file, allowing load definition to be independent of media type.

The FILES.LUM file should contain the information defined in Table 3.2.3.2. The placement of the fields within the FILES.LUM file should be as defined in Appendix C, Figure C-5, FILES.LUM File Format.

Any unused field (e.g., spare field) should be set to a bit image of all zeros.

The FILES.LUM files on all members of a media set will be identical except for the media sequence number and the FILES.LUM file CRC fields.

Table 3.2.3.2 - FILES.LUM File Content

Name of Field	Field Size (bits)	Note
FILES.LUM File Length	32	
Media File Format Version	16	
Spare	16	
Pointer to Media Set PN Length	32	
Pointer to File Number of Media Set Files	32	

Name of Field	Field Size (bits)	Note
Pointer To User Defined Data	32	
Pointer to FILES.LUM File Check Value Length	32	
Expansion Point No. 1	0	
Media Set PN Length	16	
Media Set PN	16	1
Media Sequence Number (X)	8	
No. Of Media Set Members (Y)	8	
Number of Media Set Files	16	
# File Pointer	16	
# File Name Length	16	
# File Name	16	1
# File Pathname Length	16	
# File Pathname	16	1
# File Member Sequence No.	16	
# File CRC	16	
# File Check Value Length	16	
# File Check Value Type	16	
# File Check Value	16	1
# Expansion Point No. 2	0	
Expansion Point No. 3	0	
User Defined Data	Multiples of 16	2
FILES.LUM File Check Value Length	16	
FILES.LUM File Check Value Type	16	
FILES.LUM File Check Value	16	1
FILES.LUM File CRC	16	

Notes:

1. One or more 16-bit words.
2. Zero or more 16-bit words.

# Fields are repeated as a group for each file in the media set (excluding the FILES.LUM File).

All values should be expressed as binary numbers except the noted ASCII character fields.

3.2.3.2.1 FILES.LUM File Length

The FILES.LUM File Length is the number of 16-bit words in the FILES.LUM file including this field.

3.2.3.2.2 Media File Format Version

The Media File Format Version is defined by 16-bits. The Media File Format Version is defined in Section 1.4.1, File Format Version Definition.

3.2.3.2.3 Spare

The spare field is used to align the pointers that follow, which are defined on 4-byte boundaries.

3.2.3.2.4 Pointer to Media Set PN Length

This is the absolute pointer (number of 16-bit words from start of file) to the first word of the Media Set PN Length field.

3.2.3.2.5 Pointer to Number of Media Set Files

This is the absolute pointer (number of 16-bit words from start of file) to the first word of the Number of Media Set Files field.

3.2.3.2.6 Pointer to User Defined Data

This is the absolute pointer (number of 16-bit words from start of file) to the first word of the User Defined Data field. Set to 0x0000 if there is no user defined data field.

3.2.3.2.7 Pointer to FILES.LUM Check Value Length

This is the absolute pointer (Number of 16-bit words from the start of file) to the first word of the FILES.LUM Check Value Length.

3.2.3.2.8 Expansion Point No. 1

This is the point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

3.2.3.2.9 Media Set PN Length

This is the number of characters in the Media Set PN.

This number does not include any NULs appended to fill out the field if the number of characters in the Media Set PN is odd.

3.2.3.2.10 Media Set PN

The Media Set PN is the actual Media Set PN including delimiters.

The Media Set PN field is an 8-bit ASCII character string whose length is defined by the Media Set PN Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

Implementers should ensure that the Media Set PN is compliant with the recommendations of Section 3.1.

3.2.3.2.11 Media Sequence Number (X)

This is the number of this member in the media set. Members are numbered 1 through 255. Zero (0) is not used to number members.

3.2.3.2.12 Number of Media Set Members (Y)

This is the number of media members in the media set. For a set consisting of a single member, X = 1 and Y = 1.

3.2.3.2.13 Number of Media Set Files

This is the number of files listed in the file list. All files on the media set should be included in the file list except the FILES.LUM.

3.2.3.2.14 File Pointer

The File Pointer is the relative number of 16-bit words to the next File Pointer.

The value of the File Pointer for the last File in the list should be 0x0000.

3.2.3.2.15 File Name Length

The File Name Length is the number of characters in the “File Name.”

This number does not include any NULs appended to fill out the field if the number of characters in the File Name is odd.

3.2.3.2.16 File Name

The File Name field is an 8-bit ASCII character string whose length is defined by the File Name Length field.

The File Name field should be allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append an NUL to the character string.

Implementers should ensure that the File Name is compliant with the recommendations of Section 2.2.2.

The “File Name” is the name of the file, without any information relative to its path. A File Name should never begin with a backslash nor contain any backslash. File Names.

3.2.3.2.17 File Pathname Length

The File Pathname Length is defined as the number of characters in the File Pathname field. This number does not include any NULs appended to fill out the field if the number of characters in the File Name field is odd.

3.2.3.2.18 File Pathname

The File Pathname field is an 8-bit ASCII character string whose length is defined by the File Pathname Length field.

The field should be allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

The File Pathname is the complete path to the file, without the name of the file. A Pathname should always begin at the root directory of the media member (indicated by a leading backslash). A Pathname should always finish with a backslash. When a Pathname includes one or more directory names, the Pathname is constructed with the most significant (i.e., parent) directory name first, followed by lower level (i.e., child) directory name(s). The backslash character (“\”) is used as the delimiter between concatenated directories. Files located on the top level of the media have the File Pathname field equal to “\”.

3.2.3.2.19 File Member Sequence No.

This is the number of the member in the media set that contains the subject file.

3.2.3.2.20 File CRC

The File CRC is a 16-bit CRC covering the entire file. The CRC should be calculated as defined in Section 4.3.2.

COMMENTARY

If the subject file is a header file, then the File CRC will be different than the Header File CRC embedded in the subject file. This is because the File CRC field of the FILES.LUM file includes the Header File CRC field of the subject file, whereas the Header File

CRC field of the header file excludes itself and the Load CRC field of the header file.

The calculated File CRC of the entire LOADS.LUM file will be 0x0000 because it includes the LOADS.LUM file CRC field. The calculated File CRC of the entire BATCHES.LUM file will be 0x0000 because it includes the BATCHES.LUM file CRC field. This demonstrates validity of the files and corresponding CRCs.

3.2.3.2.21 File Check Value Length

This is the number of 8-bit bytes in the File Check Value including this field and File Check Value Type. The Check Value Length should be implemented as defined in Section 5. Set value to 0x0000 when not used.

3.2.3.2.22 File Check Value Type

This indicates the type of Check Value stored in the File Check Value. The Check Value Type should be implemented as defined in Section 5. Set value to zero when not used. Omit if File Check Value Length is set to 0x0000.

3.2.3.2.23 File Check Value

This is a variable length and variable data type field containing the File Check Value. The Check Value should be implemented as defined in Section 5, consistent with the Files.Lum File Check Value type. Omit if File Check Value Length is set to 0x0000.

3.2.3.2.24 Expansion Point No. 2

This is the point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

3.2.3.2.25 Expansion Point No. 3

This is the point where file format growth may occur (new fields may be defined) in subsequent versions of the file format.

3.2.3.2.26 User Defined Data

User Defined Data is recorded in 16-bit blocks. This is an option that may be omitted. If omitted, the pointer to User Defined Data field should be set to a value of 0x0000.

3.2.3.2.27 FILES.LUM File Check Value Length

This is the number of 8-bit bytes in the Support File Check Value including this field and FILES.LUM File Check Value Type. The Check Value Length should be implemented as defined in Section 5. Value may be set to 0x0000 if not used.

3.2.3.2.28 FILES.LUM File Check Value Type

This indicates the type of Check Value stored in the FILES.LUM File Check Value. The Check Value Type should be implemented as defined in Section 5. Omit if FILES.LUM File Check Value Length is set to 0x0000.

3.2.3.2.29 FILES.LUM File Check Value

This is a variable length and variable data type field containing FILES.LUM File Check Value. The Check Value should be implemented as defined in Section 5, consistent with the FILES.LUM File Check Value type. Omit if FILES.LUM File Check Value Length is set to 0x0000.

3.2.3.2.30 FILES.LUM File CRC

The FILES.LUM File CRC is a 16-bit CRC covering the entire file, excluding the FILES.LUM File CRC field. The CRC should be calculated as defined in Section 4.

3.2.3.3 List-of-Batch File Content and Organization

The purpose of the BATCHES.LUM file is to provide an efficient access to basic information about each Optional Batch File contained on the media set.

If at least one Batch File is contained on the media set, the BATCHES.LUM should be present on each member of the media set. The BATCHES.LUM file should list every Batch File on the media set.

The BATCHES.LUM file should contain the information defined in Table 3.2.3.3-1.

Any unused field (e.g., spare field) should be set to zero.

The BATCHES.LUM file on each member of a media set should be identical except for the media sequence number and the BATCHES.LUM file CRC fields.

Table 3.2.3.3-1 - BATCHES.LUM File Content

Name of Field	Field Size (bits)	Note
BATCHES.LUM File Length	32	
Media File Format Version	16	
Spare	16	
Pointer to Media Set PN Length	32	
Pointer to Number of Batches	32	
Pointer to User Defined Data	32	
Expansion Point No. 1	0	
Media Set PN Length	16	
Media Set PN	16	1
Media Sequence Number (X)	8	
Number of Media Set Members (Y)	8	
Number of Batches	16	
+ Batch Pointer	16	
+ Batch PN Length	16	
+ Batch PN	16	1
+ Batch File Name Length	16	
+ Batch File Name	16	1
+ Member Sequence Number	16	
+ Expansion Point No. 2	0	
Expansion Point No. 3	0	
User Defined Data	Multiples of 16	2
BATCHES.LUM File CRC	16	

Notes:

1. One or more 16-bit words.
2. Zero or more 16-bit words.

+ Fields are repeated as a group for each Batch in the media set.

All values should be expressed as binary numbers except the noted for ASCII character fields.

Detailed field descriptions are listed in the following sections in the order they appear in Table 3.2.3.3-1.

3.2.3.3.1 BATCHES.LUM File Length

The BATCHES.LUM File Length is the number of 16-bit words in the BATCHES.LUM file, including this field.

3.2.3.3.2 Media File Format Version

The Media File Format Version is defined by 16-bits. The Media File Format Version is defined in Section 1.4.1, File Format Version Definition.

3.2.3.3.3 Spare

The spare field is used to align the pointers that follow, which are defined on 4-byte boundaries.

3.2.3.3.4 Pointer to Media Set PN Length

The Pointer to Media Information is defined as the absolute pointer, which is the number of 16-bit words from start of file to the Media Set PN Length field.

3.2.3.3.5 Pointer to Number of Batches

Pointer to Batch List is defined as the absolute pointer, which is the number of 16-bit words from start of file, to the first word of the “Number of Batches” field.

3.2.3.3.6 Pointer to User Defined Data

Pointer to User Defined Data is defined as the absolute pointer, which is the number of 16-bit words from start of file, to the first word of the “User Defined Data” field. The value should be set equal to 0x0000, if there is no User Defined Data field.

3.2.3.3.7 Expansion Point No. 1

Expansion Point No. 1 is the point where file format growth may occur, that is, where new fields may be defined in subsequent versions of the file format.

3.2.3.3.8 Media Set PN Length

The Media Set PN Length is the number of characters in the Media Set PN. This number does not include any NULs appended to fill out the field if the number of characters in the Media Set PN is odd.

3.2.3.3.9 Media Set PN

The Media Set PN is defined by the actual Media Set PN including delimiters.

This field contains the string of 8-bit ASCII characters representing the Media Set PN whose length is defined by the Media Set PN Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

Implementers should ensure that the Media Set PN is compliant with the recommendations of Section 3.1.

3.2.3.3.10 Media Sequence Number (X)

The Media Sequence Number (X) is defined as the number of this specific member in the media set. Members are numbered 1 through 255. Zero (0) is not used to number members.

3.2.3.3.11 Number of Media Set Members (Y)

The Number of Media Set Members (Y) is defined as the number of media members in the media set. For a set consisting of a single member, X should be set equal to 1 and Y should be set equal to 1.

3.2.3.3.12 Number of Batches

Number of Batches is defined as the number of Batch Files included in the Batch List. All batches in the media set should be included in the batch list.

3.2.3.3.13 Batch Pointer

Batch Pointer is the relative pointer, which is the number of 16-bit words to the next Batch Pointer. The value of the Batch Pointer for the last Batch in the list should be 0x0000.

3.2.3.3.14 Batch PN Length

Batch PN Length is defined as the number of characters in the Batch PN. This number does not include any NULs appended to fill out the field if the number of characters in the Batch PN is odd.

3.2.3.3.15 Batch PN

Batch PN is defined as the actual Batch PN including delimiters.

This field contains the string of 8-bit ASCII characters representing the Batch PN whose length is defined by the Batch PN Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

Implementers should ensure that the Batch PN is compliant with the recommendations of Section 2.1.1.

3.2.3.3.16 Batch File Name Length

Number of characters in the Batch File Name does not include any NULs appended to fill out the field if the number of characters in the Batch File Name is odd.

3.2.3.3.17 Batch File Name

This field contains the string of 8-bit ASCII characters representing the Batch File Name whose length is defined by the Batch File Name Length field.

The field is allocated an even number of bytes. If the number of characters to be defined in the field is odd, then append a NUL to the character string.

Implementers should ensure that the Batch File Name is compliant with the recommendations of Section 2.2.2.

The Batch File Name is the name of the file, without any information relative to its path. A Batch File Name should never begin with a backslash nor contain any backslash. Batch File Names should include all extensions and delimiters.

3.2.3.3.18 Member Sequence Number

Member Sequence Number is defined as the sequence number of the media member where the Batch File for this Batch is located.

3.2.3.3.19 Expansion Point No. 2

Expansion Point No. 2 is the point where file format growth may occur, that is, new fields may be defined, in subsequent versions of the file format. The size of the Expansion Point No. 2 should not cause the Batch Pointer to overflow.

3.2.3.3.20 Expansion Point No. 3

Expansion Point No. 3 is defined as the point where file format growth may occur, that is, new fields may be defined, in subsequent versions of the file format.

3.2.3.3.21 User Defined Data

User Defined Data is defined as an option that may be omitted. If omitted, the pointer to User Defined Data field should be set to a value of zero.

3.2.3.3.22 BATCHES.LUM File CRC

The BATCHES.LUM File CRC is defined as a 16-bit CRC covering the entire BATCHES.LUM file, excluding the BATCHES.LUM File CRC field. The CRC should be calculated as defined in Section 4.

3.2.4 Media Set File Organization

The media set files are organized by the media set creator according to these rules to allow duplicate file names within media sets.

COMMENTARY

The purpose of the file organization is to support media sets where parts have duplicate files names across parts and/or within parts. This allows support for single and multi-disk ARINC 615 parts, Boeing D6-55562-5 parts on D6-55562-6 media, as well as parts with filenames that do not completely conform to Section 2.2.2.

The media set creator should ensure that if two or more files from the same PN have duplicate file names and duplicate CRC values they also contain the same contents and may be used interchangeably.

3.2.4.1 Location of Load PN Files

All files, including the Header File, and subdirectories for a load PN should be placed in the Part Root Directory. The header file should only reference files in the Part Root Directory or in subdirectories of the Part Root Directory.

The Part Root Directory name should be unique for each load PN in a media set and is recommended to be the PN itself.

If the files for a Load PN are stored on more than one media members in a media set the identical directory name should be used on each member in the media set containing the files for the Load PN.

COMMENTARY

Files with duplicate file names may occur more than once in the subdirectories of the Part Root Directory. This is necessary to support ARINC 615 and Boeing D6-55562-5 files.

The Header File Name must be unique for each software load, as discussed earlier in Section 2.2.2, Software Load File Naming. One way to accomplish this is to embed the load PN in the Header File Name. It is recommended to use the PN of the Header File as the name of the Part Root Directory, exclusive of delimiters (“-“).

For a given software load residing on physical transport media, the Part Root Directory may be unambiguously determined as follows. First, search LOADS.LUM to obtain the Header File Name corresponding to the load PN. Second, search FILES.LUM to obtain the File Pathname for that Header File Name. This File Pathname is the Part Root Directory.

3.2.4.1.1 Storage of ARINC 615 Parts

ARINC 615 floppy disk media sets may be encoded with ARINC 665 information by adding the appropriate files to the root directory of each floppy disk in the media set.

If it is desirable to put data from multiple ARINC 615 part floppy disks on to a single ARINC 665 media set member, the following storage rule should be followed. ARINC 615 parts should be organized with the files from the first floppy disk in a directory named disk001, from the second disk in disk002 and NNNth disk in diskNNN. The diskNNN directories are stored in the Part Root Directory. The organization of the files in the diskNNN directories should conform to ARINC 615. The part Header File should be placed in the Part Root Directory.

COMMENTARY

The ARINC 665 media set containing multiple ARINC 615 parts will not itself be ARINC 615 compliant. It only provides a method of condensing multi-disk parts onto single media*.*

For more information, see ARINC Project Paper 641: *Logical Software Part Packaging for Transport.*

3.2.4.1.2 Storage of Boeing Legacy Compliant Parts

Disk Media sets designed to pre-665 Boeing specifications (Legacy), found in Boeing Documents D6-55562-5 D6-55562-6, may be encoded with ARINC 665 information by adding the appropriate files to the root directory of each floppy disk in the media set. The part header file, FILES.LUM and LOADS.LUM files should not be added to the disk.dir file.

If it is desirable to put data from multiple Boeing Legacy Part floppy disks on to a single ARINC 665 media set member, the following storage rule should be followed. Boeing Legacy parts should be organized with the files from the first disk in a directory named disk001, from the second disk in disk002 and NNNth disk in diskNNN. The diskNNN directories are stored in the Part Root Directory. The organization of the files in the diskNNN directories should conform to the ARINC 615. The part Header File should be placed in the Part Root Directory.

The calculations for the Boeing Legacy disk.dir file should be made as though the ARINC 665 directory structure did not exist. Any system moving the data from diskNNN directories will not need to recalculate the disk.dir checksums or adjust any file paths.

COMMENTARY

The ARINC 665 media set containing multiple Boeing Legacy Compliant parts will not itself be Legacy compliant. It only provides a method of condensing multi-disk parts on to single media.

3.2.4.2 Media Set Parsing Rules

Media set parsing rules allow all systems reading the media set to resolve duplicate file names within PNs and across PNs the same way.

3.2.4.2.1 Search Within Primary Root Directory

If the filename in the header file matches more than one file in FILES.LUM limit the search to files within the Part Root Directory.

3.2.4.2.2 Match File CRC Value

If the result of Section 3.2.4.2.1 returns more than one filename, use both filename and CRC value from the header file and FILES.LUM to distinguish the files.

3.2.4.2.3 Choose the First File Found in FILES.LUM

If the result of Section 3.2.4.2.2 returns more than one matching file in FILES.LUM the first file in FILES.LUM should be used.

COMMENTARY

The media set creator is responsible to ensure that both files are identical and may be used interchangeably. This rule ensures that the media set will be read in the same way on any given system.

3.2.4.3 Directory Structure for Electronic Distribution

When establishing a directory structure for a software load for electronic distribution, there are two cases depending on whether the load files have duplicate names, described as follows:

If a Loadable Software Part (LSP) is developed in compliance with this standard, all file names will be unique, per Section 2.2.2, Software Load File Naming; Section 2.2.3.1.33, Data File Name; and Section 2.2.3.1.46, Support File Name. In this case, any directory structure below the Part Root Directory is of no significance to the packaging, and may be omitted from the packaging. In effect, the method of packaging for electronic distribution may consider all files constituting the load to have been in the Part Root Directory.

If an LSP has duplicate file names in order to support ARINC 615 and Boeing Legacy formats, then (1) the content of these files, if different one from another, must be distinguishable by file CRC, and (2) the support computer file system typically requires the files with duplicated names to reside in separate subdirectories of the Root Part Directory. In this case, the method of packaging for electronic distribution should preserve the subdirectory structure required to accommodate the files with duplicated names. All other directory structure below the Part Root Directory is of no significance to the packaging, and may be omitted from the packaging. To resolve duplicate file names, it is necessary to compute the data file CRC and match the file name and CRC with the contents of the load Header File.

3.3 Media Set Labeling

3.3.1 Label Content

The **Media Set Parts** label should contain all the information defined in Table 3.3.1-1. The label may also contain the information contained in Table 3.3.1-2. Any additional information or graphics must not conflict with or hinder readability of the required information.

Information on the label should be clearly identified, e.g., the Media Set PN should be identified as follows: "Set PN: XXXXXXXXXXX." Table 3.3.1-3 provides recommendations for label information identification.

The media label content and layout should be the same for all members of the media set, except the media sequence number.

Table 3.3.1-1 – Recommended Label Content

Item	Description
1. Media set nomenclature	The title of the media set. The media set nomenclature should be composed of the target HW/LRU/System and the type (e.g., OPS, OPC, OSS, DB) of software. The title may also include the ATA Chapter.
2. Media set PN	The PN of the media set.
3. Media sequence number	Two numbers, separated by the word "of," that represent the order and total number of members in a set, e.g., XX of YY.
4. Content Description	List of the software loads that are contained on the media set. If the media contains more software loads than can be listed on the media label then the label should refer to the LOADS.LUM file for media content information.
5. Supplier Identification	The name and/or Commercial and Government Entity (CAGE) code of the company from which replacement parts can be procured.
6. Media set serial number	The unique serial number that identifies a specific media set and is the same on all members in that set.
7. Product acceptance/ release stamp	The supplier's quality control/assurance or configuration control group's stamp. The stamp should uniquely identify the supplier who owns/uses it and indicate that the LSP transport media (and its contents) have been accepted by the supplier's quality control/assurance or configuration control group(s).

Table 3.3.1-2 – Optional Label Content

Item	Description
1. Validity Date	The "use-by" date. (See Note 1)
2. Bar code	Bar code specifications are TBD. The intent is that the industry will adopt a single Bar Code Standard for use on all parts (LRUs, Media Set Parts, etc.) when such standard is adopted this document will reference it.
3. Copyright notice	The notice that information contained on the media is copyrighted.
4. Integrity Check Value	Supplier specified media integrity check type and value, as listed in section 5.
5. Label form number	A label form number that indicates a pre-printed label stock.
6. Proprietary notice	The notice that the information contained on the media is proprietary.
7. Spare parts marking	Replacement or modification part marking, per FAR 45.15.
8. Media creation date	The date that the media set was created. The date should appear in format "DD XXX YY." For example: 14 APR 98

Item	Description
9. FIN	Functional Item Number (defines the function and logical location of the item).
10. CMS	Domestic code

Note: Some media sets (e.g., FMC Nav data base) may contain information that is valid only for a specific period of time. In these cases, the label may define the time frame for which the media content is valid.

Table 3.3.1-3 – Recommended Label Information ID

Item	Recommended ID
1. Media set nomenclature	No ID required
2. Media set PN	"Set PN:"
3. Media set serial number	"Set SN:"
4. Content Description	"Software PNs:"
5. Media sequence number	"Disk x of y"
6. Media creation date	"Mfg. Date:"

3.3.2 Label Format

The media label format, color, and lay-out should be the same for all members of the media set.

The label information should be placed according to its relative importance (Tables 3.3.1-1 and 3.3.1-2 list the label information in order of relative importance). The more important information should be placed at the top of the label (when the Media Set is stowed), and be in a larger and bolder font than the less important information. For example: The Media Set nomenclature and PN could be positioned at the top of the label in bold 10-point text, whereas, the supplier identification may be placed at the bottom of the label in non-bold 6-point text.

The Media Set Nomenclature, Media Set PN and Media Set Sequence Number should be in bold text. All other information on the label should be in non-bold text.

The Media Set Nomenclature, Media Set PN and Media Set Sequence Number should be at least 10-point. The Software PN (s) may be the same size or smaller than the Media Set PN. All other information on the label should be at least 2-points smaller than the Media Set PN.

COMMENTARY

Certain label information is required for the technicians to locate and use the media to maintain the aircraft (e.g., the Media Set PN and nomenclature). Other information is required to allow the airlines to order spare (or replacement) copies of the media (e.g., the supplier's identification). It is important that the location and relative visibility of information supports the daily use of the media in maintaining the aircraft system. Key information needs to be visible when the media is stowed (i.e., in the media binder or file card box, etc.). Proprietary notice and copyright information on the label should not take precedence over software content and media identification information.

All label items should be legible and printed in indelible ink.

The media label should not reduce the life of the media.

The media label should be tamper resistant (i.e., any attempt to change label information once the label is applied to the media should be obvious).

3.4 Media Type Specific Items

The purpose of this section is to define aspects of **Media Set Parts** that are applicable to specific media types.

Inclusion of a specific media type in this section should not be considered an endorsement of its use. Suppliers should select the specific type of media to use based on the availability of readers and other criteria. For example, ARINC 615 loaders support 3.5-inch disk. ARINC 615A loaders support PC-Cards, and 3.5-inch disks and optical disks.

If a specific type of media is used the supplier should implement the following in order to ensure maximum compatibility with existing and future readers and systems.

All multi-byte words should be written to media with most significant byte first and least significant byte last. For example*:* the most significant 8-bits (MSbyte) of each 16-bit word are written to the media in the first 8-bit byte (n), followed by the least significant bits (LSbyte) in the next 8-bit byte (n+1). The same byte ordering is used to derive field information in all files on the media set.

3.4.1 Disk Sets

Each disk of a disk set should be formatted in accordance with media format specifications defined in the following Microsoft documents:

PSS ID Number: Q140418, Article last modified on 09-10-1996, detailed explanation of FAT Boot Sectors.

ISBN 1-57231-344-7, October 1996, section About File Systems. This format specification is the one used for Windows 95 and NT, allowing Long File Names capability and full downward compatibility with MS-DOS 3.1 file names.

All files should be contained in the root directory of the media member.

It is not recommended to perform concurrent parallel loads using floppy disk media.

3.4.2 PC Card

Each member of a PC Card set should conform to type 1, type 2, or type 3 form factors as defined by the "PC Card Standard" dated March 1997, including:

Volume 1:	Overview and glossary
Volume 2:	Electrical specification
Volume 3:	Physical specification
Volume 4:	Metaformat specification
Volume 5:	Card service specification
Volume 6:	Socket Services specification
Volume 7:	Media storage specification, restricted to MS-DOS FAT format supporting Long File Name (cf. ISBN 1-57231-344-7, October 1996, section about File System)
Volume 8:	PC Card ATA specification

Volume 10: Guidelines

Cartridges should be compatible with the “PC Card ATA interface standard.”

COMMENTARY

In this context, ATA refers to the ANSI AT Attachment (ATA) Interface for disk drives in the PC Card environment. ANSI is the American National Standards Institute.

All files should be contained in the first four directory levels of the media member.

3.4.3 CD-ROM

Each member of a CD set should be formatted in accordance with ISO 9660 and Joliet Long file names.

All files should be contained in the first four directory levels of the media member.

The most significant 8-bits (MSbyte) of each 16-bit word are written to the CD in the first 8-bit byte (n), followed by the least significant bits (LSbyte) in the next 8-bit byte (n+1). The same byte ordering is used for derived field information in all files on the media set.

3.4.4 Hard Disk

As an option, a hard disk can be used in the data loader or accessible on the Ethernet network. The hard disk should support the Long File Name capability, which allows full backward compatibility with MS-DOS 3.3 file names (8.3 notation).

4.0 CYCLIC REDUNDANCY CODES (CRC)

Cyclic Redundancy Codes (CRCs), also known as Cyclic Redundancy Checks, are used to detect corruption of binary data. A computation is performed on the data to yield the CRC. Whenever the data is copied or transmitted, the CRC is re-computed. If the initial and subsequent CRC values disagree, the data has been corrupted. Conversely, if the two CRCs agree, then the data probably has not been corrupted. CRC algorithms are chosen so that the chance of not detecting corrupted data is very, very small.

Section 4.1 offers CRC definition. Section 4.2 addresses rules necessary to ensure CRC computation is consistent between system environments. Section 4.3 provides parameters used in calculating CRC of variable string length.

Conventions have emerged with some variation between parameters used for each size of CRC. To retain compatibility with existing CRC generation and checking tools, the variances have been accommodated. Integrity is not compromised by retaining these differences.

Appendix K provides support materials, including a conceptual description of the CRC calculation process, followed by an example for manual calculation of a CRC. The flow for an efficient table driven CRC calculation method is also given, followed by a copy of C code producing a table driven CRC generator.

4.1 CRC Definition

The formal definition of a Cyclic Redundancy Code is described in terms of algebraic polynomials with binary coefficients. Individual bits of the data represent the coefficients of a dividend polynomial. A carefully selected nth degree divisor polynomial is used as a generator for an n-bit CRC. The CRC value is the remainder obtained after modulo 2 division of the binary data by the generator.

If the individual bits b_i of a block of binary data (L bits long) are considered to be the coefficients of a polynomial of a given variable X,

$B(X) = b_{L-1}X^{L-1} + b_{L-2}X^{L-2} + \cdots + b_1X^1 + b_0X^0$ then the n-bit CRC value of this block is the remainder (coefficient bits only) obtained from the binary division, modulo 2, of the dividend polynomial $B(X)X^n$ by an nth degree divisor polynomial,

$$G(X) = g_nX^n + g_{n-1}X^{n-1} + \cdots + g_1X^1 + g_0X^0$$

where $g_n = 1 \text{ and } g_0 = 1$.

It is important to understand that the dummy variables X never actually enter into the calculation of a CRC: only the coefficients are used. Polynomial terminology is introduced only to precisely specify the types of operations that must be performed in order to satisfy the required algebra and to unambiguously determine bit ordering.

4.2 Rules for CRC Calculation

As required in their respective sections, the Load CRC computation and Load Check Value computation both process the header file first followed by files in the order they are listed in the header file.

Within each file, bytes should be processed in the order which they occur (first byte first and last byte last).

The conceptual CRC algorithm of Appendix K has two key operations repeated in a loop: a shift, followed by a conditional Exclusive-OR operation. Unfortunately, applied algorithms are not so terse. Consideration for error detection and enhanced processing speed necessitate many adaptations. These adaptations take the following into account:

4.2.1 Bit Ordering

Each byte processed by the CRC generation tool will have its most significant (left-most) bit corresponding to the highest power of X in accordance with the definition of dividend polynomial of Section 4.1.

4.2.2 Bit Shifting

The shifting of large data blocks, as described in Appendix K must be accommodated in intervals to meet size limitations of processor shift registers.

4.2.3 Transmission Bit Reflection

In the case of a transmitted message to be CRC checked, some hardware transmits the Least Significant Bit of a byte first, some, the Most Significant Bit first. This “reverse bit ordering” is equivalent to reflecting the bits about the byte’s center: the mirror image of the normal bit order.

4.2.4 Process Bit Reflection

Some algorithms reflect the bits of their generator polynomial (with its Most Significant Bit retained) about its center before using it for CRC computations. Generally, this will not produce the same CRC value as an unreflected generator polynomial.

4.2.5 Post Process Bit Reflection

Some algorithms reflect the bits of the final computed CRC.

4.2.6 Initialization

An algorithm adhering strictly to formal CRC definition, fails to detect erroneous insertion of leading zeros. To prevent this, some algorithms initialize their CRCs to all “1” bits instead of all “0” bits. Other algorithms “one’s complement” the first n bits of the data block. These two methods produce the same CRC value as long as the data block size is greater than the CRC, e.g., for 32-bit CRCs the data length must be greater than 32 bits. For data less than the size of the CRC register, the result depends upon the chosen initialization and may vary among implementations. The implementer should pad data to at least the size of the CRC register.

4.2.7 Error Detection

One error the formal definition fails to detect is “stuck on zero,” where a transmitter continues to transmit zero bit values, regardless of the actual bits composing the data block. This can be overcome by “one’s complementing” the final CRC value prior to transmission.

4.2.8 Process Efficiency

In the algorithm of Appendix K, n bits of zero value must be shifted through the CRC partition before anything significant happens. This is inefficient. Contemporary algorithms remove this annoyance.

By applying a technique called look-ahead, CRC algorithms can eliminate the need for appending n bits of zero value to the end of the data block, thereby further improving efficiency.

An algorithm that computes a CRC value of a large file, one bit at a time, is too slow to be practical. It is possible to pre-compute a large part of the intermediate results of the computation. These pre-determined values are then placed in a table which can be rapidly accessed at run time. CRC processing speed is dramatically improved using this table-driven approach.

4.2.9 CRC Examples

Appendix K provides guidance on the calculation on Loadable Software Part CRCs.

4.3 CRC Parameters

The following tables record parameters to be applied when calculating the 8-bit, 16-bit, and 32-bit CRCs.

The tables display how the items in Section 4.2 are resolved for each CRC size.

Additionally, the tables correlate with the examples and code samples given in Appendix K.

4.3.1 8-Bit CRC

As defined in Table 4.3-1, the 8-bit CRC does not perform any pre-processing on the first input data, nor does it perform any post-processing on the CRC result. The 8-bit CRC is not intended to be used for anything besides determining check characters in the ARINC 665 formatted part numbers. The integrity of the 8-bit CRC is low enough that it should not be used anywhere else. Also, part numbers composed of printable characters will never contain leading zeros nor be all zeros as in a stuck on zero error.

Table 4.3-1 displays parameters to be applied for calculating 8-bit Loadable Software CRCs.

Table 4.3-1 – CRC-8 Parameters

Parameter	Value	Description
Width	8	The decimal width of the algorithm expressed in bits is the highest power of X in the Polynomial
Polynomial	$X^8 + 1$	The polynomial in variable X as specified by the formal definition.
Generator	0x01	Coefficients of the polynomial, Most Significant Bit suppressed, in hexadecimal form. Example: 10111 denotes coefficient of polynomial: $1X^4 + 0X^3 + 1X^2 + 1X^1 + 1X^0$. Most Significant Bit suppressed yields 0111, 07 in hexadecimal form. (The generator polynomial is not reflected)
Init	0x00	The initialization value for the CRC in hexadecimal form.
RefIn	False	If <u>True</u>: data bytes of a block are passed through the CRC algorithm with bits reflected about the mid-point. Bit 7 becomes bit 0, and converse, bit 6 becomes bit 1, and converse. If <u>False</u>: data bytes are processed without reflection.
RefOut	False	If <u>True</u>: final CRC value has bits reflected as described for “RefIn” prior to XorOut operation
XorOut	0x00	A hexadecimal value, length same as Generator that is ‘Exclusive-OR’d into the final CRC value.
AvgPro b	$2^{-8} \approx 3.9 \times 10^{-3}$	The average probability of <u>not</u> detecting corrupted data.
Check	0x00	The CRC value for a 256 byte test file with hexadecimal content of: 00 01 02 … FD FE FF

Note: This CRC does not perform and post-processing on the calculated result. Therefore, the XorOut step with a zero

value may be omitted. It should be noted that this CRC algorithym does not detect stuck at zero conditions.

4.3.2 16-Bit CRC

As defined in Table 4.3-2, the 16-bit CRC performs pre-processing on the first input data; however, it does not perform any post processing on the CRC result. The 16-bit CRC is intended to be used as a simple and quick cursory check on the correct reception of the LSAP files. It is not intended to be used as a replacement for any other checks performed naturally in transferring the LSAP files and it is not intended to be used as the only validation check on LSAPs stored in NVM. The 16-bit CRC does not have high enough integrity and due to the limitation on the data size upon which it can reliably detect errors, the CRC-16 algorithm should not be used as a replacement for other checks or LSAP validation.

Table 4.3-2 displays parameters to be applied for calculating 16-bit Loadable Software CRCs.

Table 4.3-2 – CRC-16 Parameters

Parameter	Value	Description
Width	16	The decimal width of the algorithm expressed in bits is the highest power of X in the Polynomial
Polynomial	$X^{16}+X^{12}+X^5+1$	The polynomial in variable X as specified by the formal definition.
Generator	0x1021	Coefficients of the polynomial, Most Significant Bit suppressed, in hexadecimal form. Example: 10111 denotes coefficient of polynomial: $1X^4 + 0X^3 + 1X^2 + 1X^1 + 1X^0$. Most Significant Bit suppressed yields 0111, 07 in hexadecimal form. (The generator polynomial is not reflected)
Init	0xFFFF	The initialization value for the CRC in hexadecimal form.
RefIn	False	If <u>True</u>: data bytes of a block are passed through the CRC algorithm with bits reflected about the mid-point. Bit 7 becomes bit 0, and converse, bit 6 becomes bit 1, and converse. If <u>False</u>: data bytes are processed without reflection.
RefOut	False	If <u>True</u>: final CRC value has bits reflected as described for “RefIn” prior to XorOut operation
XorOut	0x0000	A hexadecimal value, length same as Generator that is ‘Exclusive-OR’d into the final CRC value.
AvgPro b	$2^{-16}\approx1.53\times10^{-5}$	The average probability of <u>not</u> detecting corrupted data.
Check	0x3FBD	The CRC value for a 256 byte test file with hexadecimal content of: 00 01 02 … FD FE FF

4.3.3 32-Bit CRC

Table 4.3-3 displays parameters to be applied for calculating 32-bit Loadable Software CRCs.

Table 4.3-3 – CRC-32 Parameters

Parameter	Value	Description
Width	32	The decimal width of the algorithm expressed in bits is the highest power of X in the Polynomial
Polynomial	$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$	The polynomial in variable X as specified by the formal definition.

Parameter	Value	Description
Generator	0x04C11DB7	Coefficients of the polynomial, Most Significant Bit suppressed, in hexadecimal form. Example: 10111 denotes coefficient of polynomial $1X^4 + 0X^3 + 1X^2 + 1X^1 + 1X^0$. Most Significant Bit suppressed yields 0111, 07 in hexadecimal form. (The generator polynomial is not reflected)
Init	OxFFFFFFFF	The initialization value for the CRC in hexadecimal form.
RefIn	False	If True: data bytes of a block are passed through the CRC algorithm with bits reflected about the mid-point. Bit 7 becomes bit 0, and converse, bit 6 becomes bit 1, and converse. If False: data bytes are processed without reflection.
RefOut	False	If True: final CRC value has bits reflected as described for “RefIn” prior to XorOut operation
XorOut	0xFFFFFFFF	A hexadecimal value, length same as Generator that is ‘Exclusive-OR’d into the final CRC value. (This step effectively ‘ones complements’ bits of final CRCs.)
AvgPro b	$2^{-32} \approx 2.33 \times 10^{-10}$	The average probability of not detecting corrupted data.
Check	0xB6B5EE95	The CRC value for a 256 byte test file with hexadecimal content of: 00 01 02 … FD FE FF

4.4 CRC Conventions

4.4.1 CRC Self Reflection

For instances where a CRC value is recorded within its subject domain, the field reserved to house the CRC value, is not taken into account during calculation. The file is processed as if data on either side of the CRC reserve are sequential.

Instructions for calculating specific CRCs should be specified in the document in which the specific CRC is named.

4.4.2 File Size Limitations

CRCs provide corruption detection with reasonable probability, for files within a specified range size. The CRC-8 is sufficient for its application against the part number. The range for CRC-16 includes files under 32,751 bits, which is approximately 4093 bytes. The range for CRC-32 includes files less than 512 Mbytes. These measures represent generally accepted ranges, because the exact point of integrity loss may be altered based on content and structure of the file. The rate at which integrity degrades also varies. Accordingly, the 512 Mbytes is a reasonable, conservative gauge.

Files crossing these thresholds should be judicially segmented into distinct files of acceptable size, at the supplier’s discretion.

5.0 INTEGRITY CHECK METHODS

5.1 Integrity Check Methods

The term Check Value includes CRC and other calculations which produce a deterministic output used for data validation. ARINC Report 665 provides for optional variable length check values for data and support files. In cases where more than one Check Value or CRC may be used the following table provides for identification of the method used. All options for 665 files are enumerated in Section 5.2. Section 5.3 provides references to further check value calculations beyond those found in Section 4.0.

Each variable length check value is specified with three parameters: an 8-bit length of the data check value in 8-bit words, 8-bit type and the calculation result. **Up to 255 check values may be enumerated with this method.**

Check values are stored in "Big Endian" format, aligned on 16-bit words. The highest ordered bytes are therefore stored at the lower address (first) in the file.

~~The length value stored in the ARINC 665 files should be set to 001xX00006 if no check value is specified and to include the length of the type when it is included.~~

COMMENTARY:

Even though the table enumerates currently specified check methods, future check methods may be defined. The length value is included to allow software, which does not recognize an enumerated type, to move on to data fields that follow the check value.

~~The type value should be set to 0x00~~$_{16}$ ~~if no check value is specified. Up to 255 check values may be enumerated with this method.~~

~~Check values are stored in "Big Endian" format, aligned on 16-bit words. The highest ordered bytes are therefore stored at the lower address (first) in the file and a check value that only requires an odd number of bytes for storage should be preceded by a 0x0000~~$_{16}$ ~~byte.~~

5.2 Data Check Value Enumeration

The 8-bit CRC check value type enumeration (Type 1) should not be used within LSPs. This 8-bit CRC is used in the part number of LSPs and BFPs only. The 8-bit CRC does not provide adequate error detection for anything larger that the LSP of BFP part numbers.

In Table 5.1, the columns contain the following data:

- Type: The ARINC 665 Check Value Enumerations.
- Title: Section title for the Check Value.
- Section: Number of the referenced ARINC 665 Section, which defines the check value calculation.
- Length in 8-bit Words: The length of the check value calculation result.

Table 5.1 - Check Value Enumeration

Type	Title	Reference Section of this Report	Length in Bytes
1	8-bit CRC	4.3.1	2
2	16-bit CRC	4.3.2	2
3	32-bit CRC	4.3.3	4
4	MD5	5.3.1	16
5	SHA-1	5.3.2	20

5.3 Integrity Check Type

In addition to the ARINC 665 CRC methods of Section 4.0, the following are valid Integrity Check Methods.

5.3.1 Message Digest (MD) 5 Integrity Check

The implementer should implement the MD5 algorithm as defined by the IETF RFC 1321 – The MD5 Message Digest Algorithm.

5.3.2 Secure Hash Algorithm-1 (SHA-1) Integrity Check

The implementer should implement the SHA-1 value of the data as specified by ANSI X9.30 (part 2).

ATTACHMENT 1 MANUFACTURER'S CODE ASSIGNMENTS

This attachment refers to a list of assigned Manufacturer's Codes (MMM). Organizations may request a Manufacturer's Code from the ARINC Industry Activities staff by submitting information requested in Appendix J.

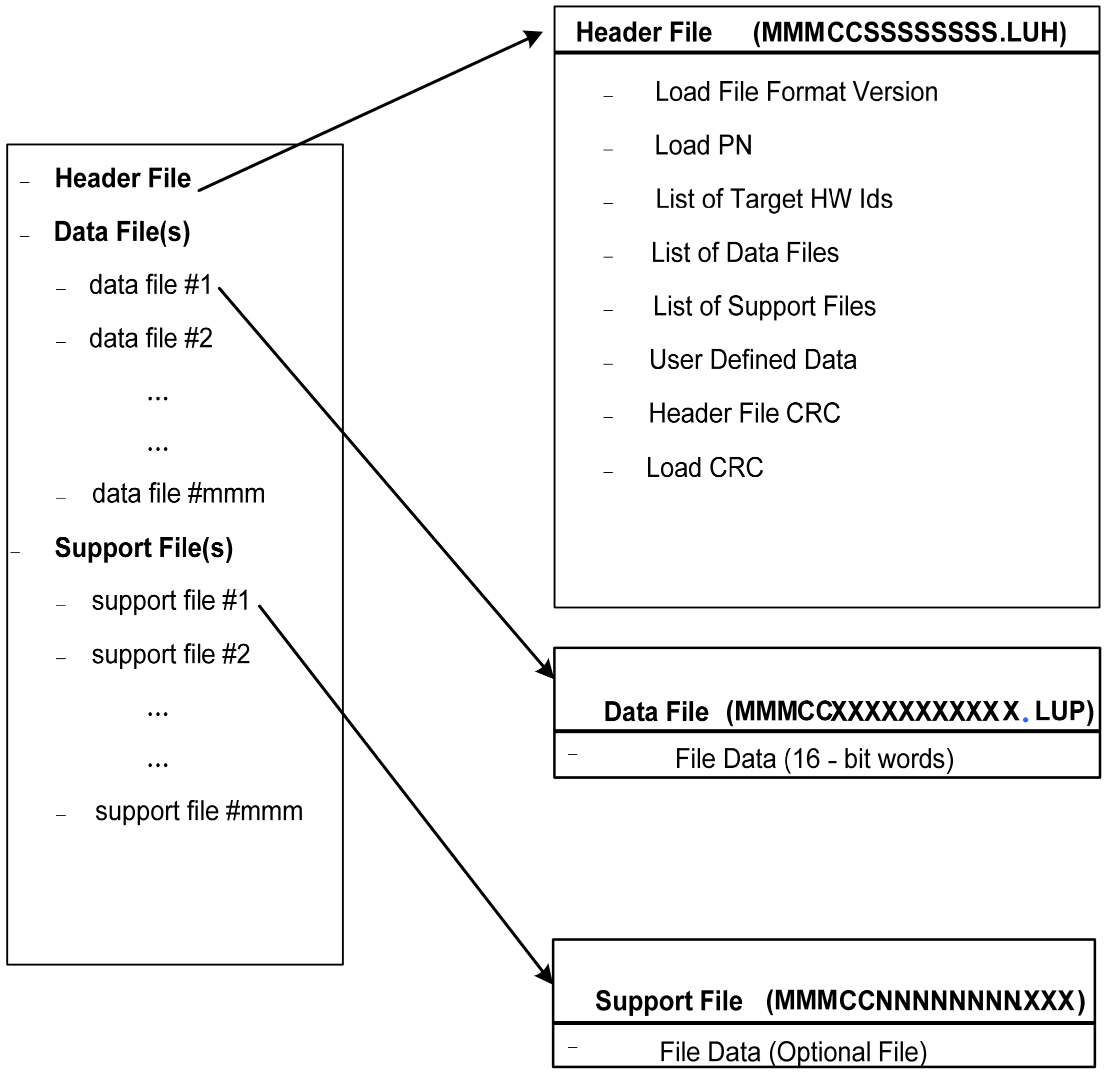
The Manufacturer's Code list includes software suppliers and airlines. It is recognized that airlines may be suppliers of software programs. Please note that the manufacturer code for an airline may be different from their ICAO identification, appearing in parenthesis after the airline name.

This list of Manufacturer's Codes is provided in a **published MMM List**, which is available from the ARINC Website:

http://www.aviation-ia.com/aeec/projects/manufacturer_code/index.html

The **MMM List** will be periodically updated with newly assigned **or changed** MMM Code Identifiers.

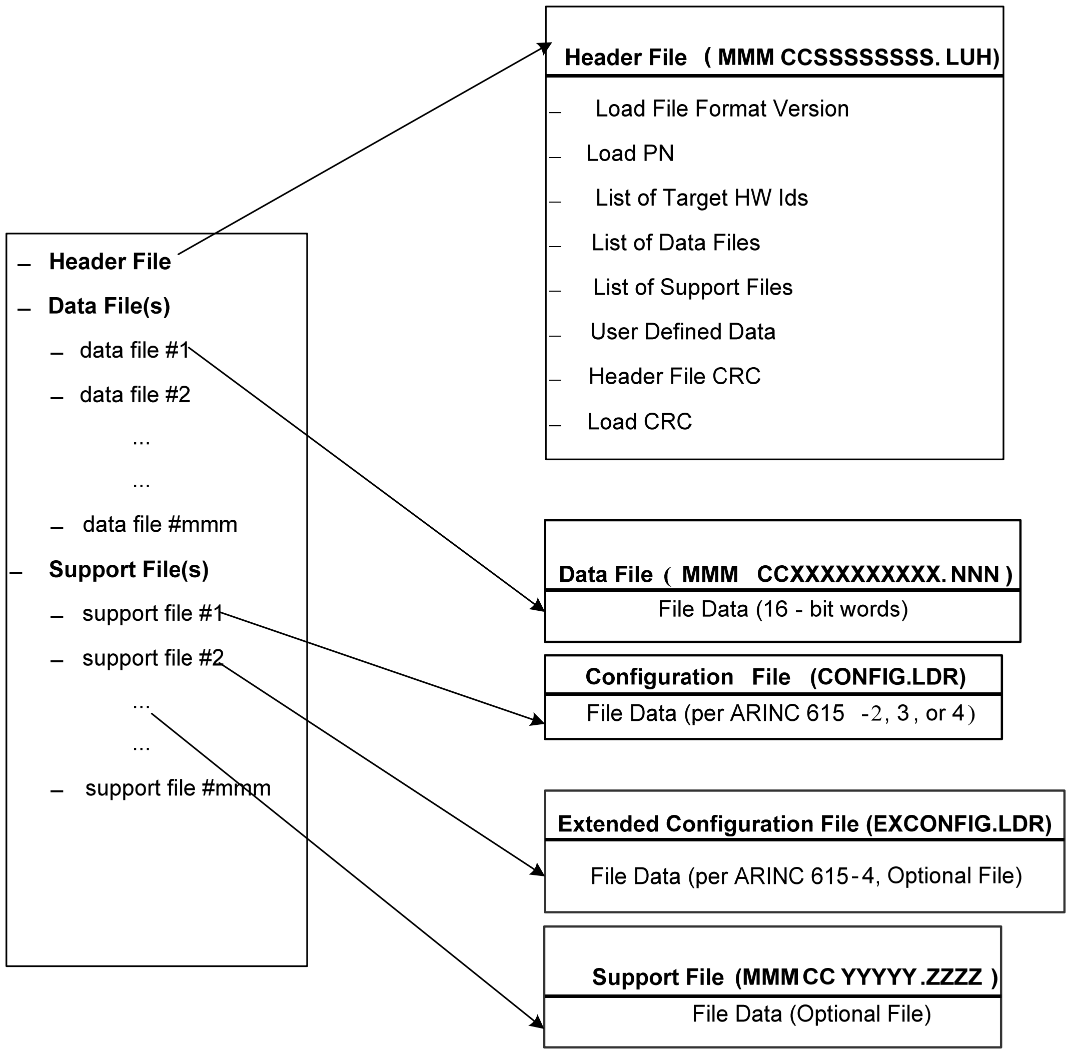
APPENDIX A LOAD STRUCTURE



(Length and content of N, S, X, Y and Z variable strings are assigned at supplier's discretion)

Figure A-1 – Structure of Typical ARINC 665 Load

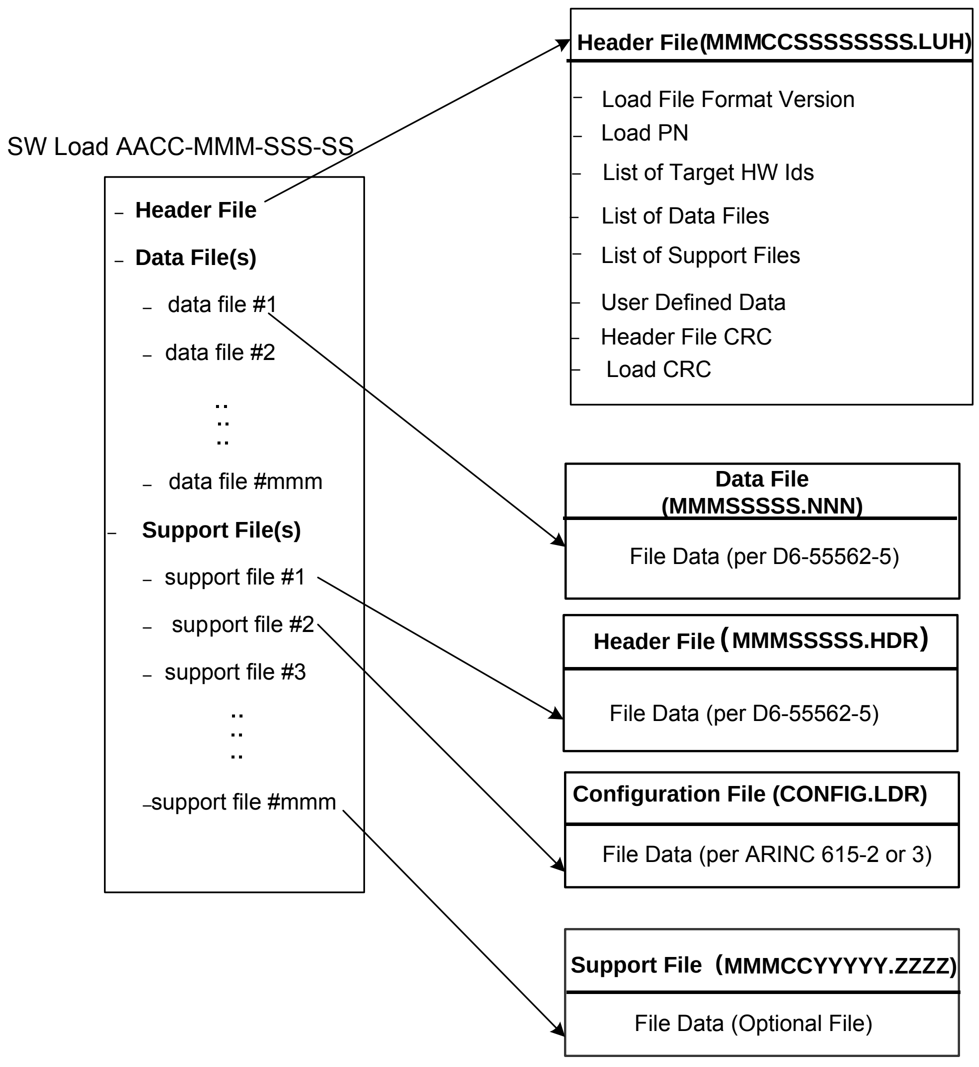
APPENDIX A
LOAD STRUCTURE


Header File (MMM CCSSSSSSSS. LUH)
– Load File Format Version
– Load PN
– List of Target HW Ids
– List of Data Files
– List of Support Files
– User Defined Data
– Header File CRC
– Load CRC
– Header File
– Data File(s)
– data file #1
– data file #2
...
...
– data file #mmm
– Support File(s)
– support file #1
– support file #2
...
...
– support file #mmm
Data File (MMM CCXXXXXXXXXX. NNN)
File Data (16 - bit words)
Configuration File (CONFIG.LDR)
File Data (per ARINC 615 -2, 3, or 4)
Extended Configuration File (EXCONFIG.LDR)
File Data (per ARINC 615-4, Optional File)
Support File (MMM CC YYYYY .ZZZZ)
File Data (Optional File)

(N, S, X, Y, and Z values are assigned at supplier's discretion)

Figure A-2 – Structure of ARINC 615 Compatible Load

APPENDIX A
LOAD STRUCTURE



(Length and content of N, S, X, Y, and Z variable strings are assigned at supplier's discretion)

Figure A-3 – Structure of ARINC 615 and ARINC 629 (Boeing 777) Compatible Load

APPENDIX B MEDIA SET STRUCTURE

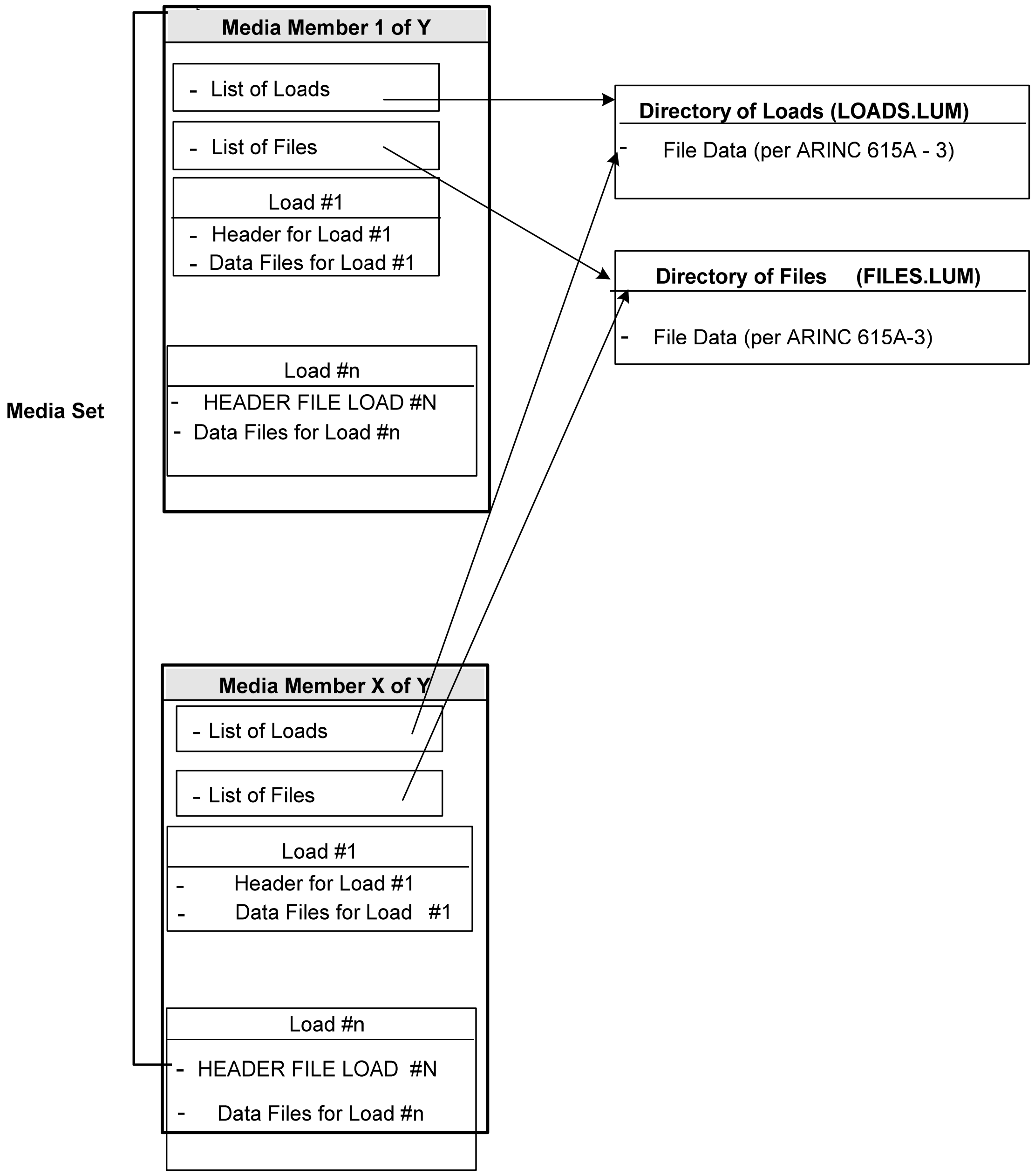


Figure B-1 – Standard Media Set Structure

APPENDIX C FILE FORMATS

C-1Header File Format

	MSB Header File —MMMSSSSSSSS.LUH	**LSB**
	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
	Header File Length (Most Significant Word)	
	Header File Length (Least Significant Word)	
	Load File Format Version	
	Part Flags	
	Pointer to Load PN **Length** (Most Significant Word)	
	Pointer to Load PN **Length** (Least Significant Word)	
	Pointer to Number of Target HW ID***s*** (Most Significant Word)	
	Pointer to Number of Target HW ID***s*** (Least Significant Word)	
	Pointer to Number of Data File***s*** (Most Significant Word)	
	Pointer to Number of Data File***s*** (Least Significant Word)	
	Pointer to Number of Support File***s*** (Most Significant Word)	
	Pointer to Number of Support File***s*** (Least Significant Word)	
	Pointer to User Defined Data (Most Significant Word)	
	Pointer to User Defined Data (Least Significant Word)	
	Pointer to Load Type Description **Length** (Most Significant Word)	
	Pointer to Load Type Description **Length** (Least Significant Word)	
	Pointer to Number of Target HW ID with Positions (Most Significant Word)	
	Pointer to Number of Target HW ID with Positions (Least Significant Word)	
	Pointer to Load Check Value Length (Most Significant Word)	
	Pointer to Load Check Value Length (Least Significant Word)	
	Load PN Length	
	Load PN (MSByte)	Load PN (MSByte-1)
	●●●	●●●
	LSByte if Load ***PN*** Length odd or LSByte+1 if Load PN Length even	NUL if Load PN length odd or LSByte if Load ***PN*** Length even
	Load Type Description Length	
	Load Type Description (MSByte)	Load Type Description (MSByte-1)
	●●●	●●●
	LSByte if Load Type Description Length odd or LSByte+1 if Load Type Description Length even	NUL if Load Type Description Length odd or LSByte if Load Type Description Length even
	Load Type ID	
	Number of Target HW IDs	
*	Target HW ID Length	
*	Target HW ID (MSByte)	Target HW ID (MSByte-1)
*	●●●	●●●
*	LSByte if Target HW ID Length odd or LSByte+1 if TargetHW ID Length even	NUL if Target HW ID Length odd or LSByte if Target HW ID Length even
	Number of Target HW ID with Position	
%	Target HW ID **with Positions** Length	
%	Target HW ID **with Positions** (MSByte)	Target HW ID **with Positions** (MSByte-1)
%	●●●	●●●
%	LSByte if Target HW ID **with Positions** Length odd or LSByte+1 if TargetHW ID **with Positions** Length even	NUL if Target HW ID **with Positions** Length odd or LSByte if Target HW ID **with Positions** Length even
%&	Number of Positions	
%&	Position Length	

APPENDIX C
FILE FORMATS

%&	Position (MSByte)	Position (MSByte-1)
%&	●●●	●●●
%&	LSByte if Position Length odd or LSByte+1 if Position Length even	NUL if Position Length odd or LSByte Position Length even
	Number of Data Files	
+	Data File Pointer	
+	Data File Name Length	
+	Data File Name (MSByte)	Data File Name Byte (MSByte-1)
+	●●●	●●●
+	LSByte if Data File Name Length odd or LSByte+1 if DataFile Name Length even	NUL if Data File Name Length odd or LSByte if DataFile Name Length even
+	Data File PN Length	
+	Data File PN (MSByte)	Data File PN Byte (MSByte-1)
+	●●●	●●●
+	LSByte if Data File PN Length odd or LSByte+1 if DataFile PN Length even	NUL if Data File PN Length odd or LSByte if Data File PN Length even
+	Data File Length (Most Significant Word)	
+	Data File Length (Least Significant Word)	
+	Data File CRC	
+	Data File Length in Bytes (Most Significant Word)	
+	Data File Length in Bytes (Most-1 Significant Word)	
+	Data File Length in Bytes (Least+1 Significant Word)	
+	Data File Length in Bytes (Least Significant Word)	
+	Data File Check Value Length	
+	Data File Check Value Type	
+	Data File Check Value (MSByte)	Data File Check Value (MSByte-1)
+	●●●	●●●
+	Data File Check Value (LSByte+1)	Data File Check Value (LSByte)
	Number of Support Files	
#	Support File Pointer	
#	Support File Name Length	
#	Support File Name (MSByte)	Support File Name (MSByte-1)
#	●●●	●●●
#	LSByte if Support File Name Length odd or LSByte+1 if Support File Name Length even	NUL if Support File Name Length odd or LSByte if Support File Name Length even
#	Support File PN Length	
#	Support File PN (MSByte)	Support File PN (MSByte-1)
#	●●●	●●●
#	LSByte if Support File PN Length odd or LSByte+1 if Support File PN Length even	NUL if Support File PN Length odd or LSByte if Support File PN Length even
#	Support File Length (Most Significant Word)	
#	Support File Length (Least Significant Word)	
#	Support File CRC	
#	~~Support File Length in Bytes (Most Significant Word)~~	
#	~~Support File Length in Bytes (Most-1 Significant Word)~~	
#	~~Support File Length in Bytes (Least+1 Significant Word)~~	
#	~~Support File Length in Bytes (Least Significant Word)~~	
#	Support File Check Value Length	
#	Support File Check Value Type	
#	Support File Check Value (MSByte)	Support File Check Value (MSByte-1)
#	●●●	●●●
#	Support File Check Value (LSByte+1)	Support File Check Value (LSByte)
	User Defined Data	

●●●	
User Defined Data	
Load Check Value Length	
Load Check Value Type	
Load Check Value (MSByte)	Load Check Value (MSByte-1)
●●●	●●●
Load Check Value (LSByte+1)	Load Check Value (LSByte)
Header File CRC	
Load CRC (Most Significant Word)	
Load CRC (Least Significant Word)	

Notes: Bold horizontal lines indicate the position of expansion points.

*	Fields repeated as a group for each Target HW ID.
%	Fields repeated as a group for each Target HW ID with Positions.
&	Fields repeated as a group for each Position within a Target HW ID with Positions group.
+	Fields repeated as a group for each Data File.
#	Fields repeated as a group for each Support File. If no support files are included in the load, then these fields are omitted.

Figure C-1 – Header File Format

C-2Data File Format

The format of the data file content is up to the LSP supplier, with the single exception that each data file should contain an integral number of 8-bit bytes. **An odd number of bytes is permitted.** ~~unless the Data File Length in Bytes field is used.~~

C-3Support File Format

The format of the support file content is up to the supplier of the software load, with the single exception that each support file should contain an integral number of 8-bit words. Note: If the ARINC 615 protocol is used for loading, then the ARINC 615-2 or later defined CONFIG.LDR file should be included as a support file of the load.

C-4LOADS.LUM File Format

	MSB		**List-of-Loads File — LOADS.LUM**													**LSB**
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	LOADS.LUM File Length (Most Significant Word)															
	LOADS.LUM File Length (Least Significant Word)															
	Media File Format Version															
	Spare															
	Pointer to Media Information (Most Significant Word)															
	Pointer to Media Information (Least Significant Word)															
	Pointer to Load List (Most Significant Word)															
	Pointer to Load List (Least Significant Word)															
	Pointer to User Defined Data (Most Significant Word)															
	Pointer to User Defined Data (Least Significant Word)															
	Media Set PN Length															
	Media Set PN (MSByte)								Media Set PN (MSByte-1)							
	●●●								●●●							
	LSByte if Media Set PN length odd or LSByte+1 if Media Set PN length even								NUL if Media Set PN length odd or LSByte if Media Set PN length even							
	Media Sequence Number (X)								No. Of Media Set Members (Y)							
	Number of Loads															
+	Load Pointer															
+	Load PN Length															
+	Load PN (MSByte)								Load PN (MSByte-1)							
+	●●●								●●●							
+	LSByte if Load PN length odd or LSByte+1 if Load PN length even								NUL if Load PN length odd or LSByte if Load PN length even							
+	Header File Pathname Length															
+	Header File Pathname (MSByte)								Header File Pathname (MSByte-1)							
+	●●●								●●●							
+	LSByte if Header File Pathname length odd or LSByte+1 if Header File Pathname length even								NUL if Header File Pathname length odd or LSByte if Header File Pathname length even							
+	Member Sequence Number															
+	Number of Target HW IDs															
+*	Target HW ID Length															
+*	Target HW ID (MSByte)								Target HW ID (MSByte-1)							
+*	●●●								●●●							
+*	LSByte if Target HW ID length odd or LSByte+1 if Target HW ID length even								NUL if Target HW ID length odd or LSByte if Target HW ID length even							
	User Defined Data															
	●●●															
	User Defined Data															
	LOADS.LUM File CRC															

Notes: Bold Horizontal lines indicate the position of expansion points.

+ Words are repeated as a group for each load in the media set.

* Words are repeated as a group for each Target HW ID defined for the load.

Figure C-4 - LOADS.LUM File Format

APPENDIX C
FILE FORMATS

C-5FILES.LUM File Format

	MSB List-of-Files File — FILES.LUM	LSB
	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
	FILES.LUM File Length (Most Significant Word)	
	FILES.LUM File Length (Least Significant Word)	
	Media File Format Version	
	Spare	
	Pointer to Media Information (Most Significant Word)	
	Pointer to Media Information (Least Significant Word)	
	Pointer to File List (Most Significant Word)	
	Pointer to File List (Least Significant Word)	
	Pointer to User Defined Data (Most Significant Word)	
	Pointer to User Defined Data (Least Significant Word)	
	Pointer to FILES.LUM File Check Value Length(Most Significant Word)	
	Pointer to FILES.LUM File Check Value Length(Least Significant Word)	
	Media Set PN Length	
	Media Set PN (MSByte)	Media Set PN (MSByte-1)
	●●●	●●●
	LSByte if Media Set PN length odd or LSByte+1 if Media Set PN length even	NUL if Media Set PN length odd or LSByte if Media Set PN length even
	Media Sequence Number (X)	Number of Media Set Members (Y)
	Number of Media Set Files	
#	File Pointer	
#	File Pathname Length	
#	File Pathname (MSByte)	File Pathname (MSByte-1)
#	●●●	●●●
#	LSByte if File Pathname length odd or LSByte+1 if File Pathname length even	NUL if File Pathname length odd or LSByte if File Pathname length even
#	File Member Sequence No.	
#	File CRC	
#	File Check Value Length	
#	File Check Value Type	
#	File Check Value (MSByte)	File Check Value (MSByte-1)
#	●●●	●●●
#	File Check Value (LSByte+1)	LSByte File Check Value (LSByte)
	User Defined Data	
	●●●	
	User Defined Data	
	FILES.LUM File Check Value Length	
	FILES.LUM File Check Value Type	
	FILES.LUM File Check Value (MSByte)	FILES.LUM File Check Value (MSByte-1)
	●●●	●●●
	FILES.LUM File Check Value (LSByte+1)	FILES.LUM File Check Value (LSByte)
	FILES.LUM File CRC	

Note: Bold Horizontal lines indicate the position of expansion points.

# Words are repeated as a group for each file in the media set (excluding the FILES.LUM File).

Figure C-5 - FILES.LUM File Format

APPENDIX D EXAMPLES

Content to Appendix D is withdrawn. Appendix D is Reserved.

APPENDIX E MANUAL METHOD FOR CALCULATING THE “CC” VALUE

The Software Part Number CC field characters can be either computed as defined in Section 4 or can be manually computed using the follow method. Both methods create the same CC characters.

The six-step procedure, with an example for each step, follows:

Step 1:
Establish the characters for the PN before the check characters are known:

ACM??-1234-5678 (?? denoting unresolved CC values, not included in the calculation)

Step 2:
Exclude delimiters and the unresolved CC values, resulting in: ACM12345678

Step 3:
Convert the ASCII characters to hexadecimal and binary equivalent:

“A”	= 0x41 = 0100 0001
“C”	= 0x43 = 0100 0011
“M”	= 0x4D= 0100 1101
“1”	= 0x31 = 0011 0001
“2”	= 0x32 = 0011 0010
“3”	= 0x33 = 0011 0011
“4”	= 0x34 = 0011 0100
“5”	= 0x35 = 0011 0101
“6”	= 0x36 = 0011 0110
“7”	= 0x37 = 0011 0111
“8”	= 0x38 = 0011 1000

Step 4:
Add the binary equivalent characters using mod 2 addition rules (0+0=0, 0+1=1, 1+0=1, 1+1=0, No carry):

sum = 0100 0111

Step 5:
Express the resulting value in upper case hexadecimal characters:

0x47 => “47”

Step 6:
Construct the final PN, including delimiters:

ACM47-1234-5678

APPENDIX F IMPLEMENTATION FOR MULTI-STANDARD COMPATIBILITY

This appendix provides guidance for creating LSP and media sets compatible with deployed ARINC 615 (Supplement 2 and later) and Boeing Legacy standards (Pre-ARINC 665 D6-55562-5 and -6 documents). This section does not supplant guidance provided by the ARINC 615 (Supplement 2 and later) or Boeing Legacy specifications. Compliant media sets can only be constructed possessing the knowledge contained in those documents as applicable.

COMMENTARY

Cross platform operation is only possible if the media, file system, and file names meet the applicable requirements of ARINC 615 or Boeing Legacy standards.

Multi-standard LSPs are not yet common and have not been implemented in all support tools and data loaders consistently. Any multi-standard LSP should be tested for compatibility with representative models of fielded tools before distribution.

At the time of Supplement 3 to this publication (2005), Boeing reported that Boeing Standards have been aligned to recognize both ARINC 665 and Legacy LSP and Media Set structures. Systems incorporating legacy designs must be carefully structured to accommodate Load packaging, receiving, storage, management, and deployment tools. The following offers guidance to accommodate transition LSPs having to accommodate Boeing Legacy and 665 standards.

Compliance with Boeing Legacy and ARINC 665 standards will not enable the spanning of an LSP across multiple media members.

Single diskette ARINC 615 (Supplement 2 and later) or Boeing Legacy compatible media sets are easily accommodated by the ARINC 665 file structure. For multi-diskette media sets, observe the following restrictions:

- The ARINC 665 file creation tool should accommodate duplicate filenames within a part and across the media set.
- The ARINC 665 verification process should be able to identify files by the combination of filename and CRC.
- Loaders and targets should support TFTP options specifying the file name with its associated CRC

F-1 ARINC 615 (Supplement 2 and Later) and ARINC 615A

This section defines how to create loads and media sets that are compatible with both ARINC 615-2/3/4 and ARINC 615A Loaders.

F-1.1 Construction of Media Set and Files

First the media set and files should be constructed following requirements of ARINC 615 (Supplement 2 and later) and observing requirements therein.

F-1.2 Addition of Files to the Media Set

Second, the required ARINC 665 files should be added to the media set without modifying the files created in F.1.1.

- The CONFIG.LDR and/or EXCONFIG.LDR file(s) should be listed in the support file section of the part header file.
- All other files created in F.1.1 should be listed in as data file section of the part header file.

F-2 Boeing 777, ARINC 629, and ARINC 615A

This section defines how to create loads and media sets that are compatible with both ARINC 615A loaders and the Boeing 777 loader (ARINC 629).

F-2.1 Construction of the Load

First, the load should be constructed following Boeing legacy specifications (D6-55562-5), including the defined PN format and Header/Data file naming rules.

F-2.2 Creation of the Files

Second, the ARINC 665 compatible files should be created as defined in Section 2, with the following guidance:

- Files created in Section F.2.1 should not be modified in any manner.
- The D6-55562-5 Header File should be listed as a Support File in the ARINC 665 part Header File.
- The D6-55562-5 Data Files should be listed as Data Files in the ARINC 665 Header File.

F-2.3 Creation of the Media Set

Third, create the media set.

- LOADS.LUM and FILES.LUM should be as defined in Section 3 of this standard
- DISK.DIR and NON_LOAD.CRC files should be constructed for each member of the media set as defined in D6-55562-6.
- The DISK.DIR and NON_LOAD.CRC files should be listed in the FILES.LUM but should not be listed in either the ARINC 665 Header File or the D6-55562-5 Header File because they are not component parts of any load.

F-2.4 Exceptions for the Load Media Set

The above procedure will create a load media set that is fully compliant with the Boeing Legacy specifications, 777 (ARINC 629), and fully compatible with ARINC 615A loader standard and compliant with the ARINC 665 loadable software standards except as follows:

- The load part number will not conform to ARINC 665, Section 2.1 defined format.

F-3 Boeing 777 (ARINC 629) and ARINC 615 (Supplement 2 and Later) and ARINC 615A

This section defines how to create loads and media sets that are mutually compatible with ARINC 615A loaders, ARINC 615 (Supplement 2 and later) loaders, and the Boeing 777 (ARINC 629) Data Load System.

- The load and media should be constructed as defined in Section F.2 above, with the ARINC 615 files listed in the part header file as defined in Section F.1.2.

APPENDIX G ACRONYMS AND ABBREVIATIONS

ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
ATA	Airlines For America
ATA	ANSI AT Attachment
BPH	Bit Pattern Header
CAGE	Commercial and Government Entity
COTS	Commercial Off The Shelf
CRC	Cyclic Redundancy Check (Code)
DB	Data Base
DLS	Data Load System
DOS	Disk Operating System
EASA	European Aviation Safety Agency
EDS	Electronic Distribution of Software
FAA	Federal Aviation Administration
FAR	Federal Airworthiness Regulation
HW	Hardware
ID	Identification/Identifier
IMA	Integrated Modular Avionics
Kbyte	Kilo bytes, 1024 bytes
LRU	Line Replaceable Unit
LSAP	Loadable Software Airplane (Aircraft) Part
LSB	Least Significant Bit
LSP	Loadable Software Parts
LSbyte	Least Significant Byte
MAT	Maintenance Access Terminal
Mbyte	Megabyte, 1,048,576 bytes
MSB	Most Significant Bit
MSbyte	Most Significant Byte
NDB	Navigation Data Base
OEM	Original Equipment Manufacturer
OSS	Option Selectable Software
OPC	Operational Program Configuration
OPS	Operational Program Software
PMAT	Portable Maintenance Access Terminal
PN	Part Number
SAL	System Address Label
SW	Software

APPENDIX H LOADABLE SOFTWARE TERMINOLOGY

Airline Modifiable Information (AMI)

Software Loads generated by the airlines to customize system operations.

Boeing Legacy Part

Prior to the formation of ARINC Report 665, Boeing had established standards for software parts. LSP standards were found in Boeing Document D6-55562-5. Media Set Part standards were in Document D6-55562-6. Parts designed in compliance with these early Boeing standards, are referred to as Boeing Legacy parts.

Although the intent of pre-ARINC 665 Boeing Standards compliments that of ARINC Report 665, the structures are not directly compatible. Accordingly, Boeing Legacy parts may be represented in ARINC 665 LSP and Media Set Part formats using select provisions, found in attachments to ARINC Report 665.

Boot Software (Boot SW)

A program used for starting the computer, which usually clears memory, sets up I/O devices, and loads the operating system. For software loading purposes, the boot is the minimum software that must be present to load software parts into the target hardware.

Check Value

Cyclic Redundancy Codes have traditionally been used to validate instances of an LSP. As file sizes and technology options grow, alternatives options to ensure integrity become optimal. Check Value fields enable LSPs to apply advanced integrity options.

Common

A level of “sameness” that invokes familiarity to the point that no additional instructions or training is required when dealing with any member of the “common” set.

Configuration Control

The process of recording, evaluating, approving or disapproving and coordinating changes to configuration items after formal establishment of their configuration identification or to baselines after their establishment.

The systematic evaluation, coordination, approval or disapproval, and implementation of approved changes in the configuration of a configuration item after formal establishment of its configuration identification or to based lines after their establishment.

Cross Load

The act or ability to load a target hardware from an already loaded target hardware, generally of the same type.

Cross Unit

Generally referring to another target hardware of the same type in a multi-target hardware installation (e.g., the left FMC is the cross unit of the right FMC).

Sometimes it may refer to other target hardware of the same system (e.g., control panel and computer).

Cyclic Redundancy Check/Code (CRC)

A value calculated from a block of data and used to detect changes to the data due to, for example, corruption of memory. CRC algorithms are chosen so that changes in the block of data are very likely to change the calculated value.

Data Base (DB)

A systematic organization of data, which facilitates access, retrieval and update.

Data File

A specific file that contains, in addition to other information, the actual data that is the object of the load process. One or more data files plus a header file make up a load. See Section 2.2.3.2 for content and format.

Data Load System (DLS)

The system on the aircraft which is used for loading. The system includes the load source, load control function, transfer medium and the target hardware. Components of a DLS may include: ARINC 615 loader, MAT, Gatelink, AIMS DLGF, bus to the target hardware, etc.

Dataloader (Software Loader)

Equipment (hardware and software) used to upload or download software (e.g., MAT, PMAT, ARINC 615 loader, etc.).

Data Loading

See “software loading.”

Deviation

The formal acknowledgment and documentation that a specific requirement will not be implemented.

Disk

A 3.5-inch Flexible Disk Cartridge as specified in ISO/IEC 9529-1 “International Standard - Dimensions, Physical and Magnetic Characteristics” Section 7.1.

Download (Down Load)

Refers to data transfer from a system to a transport or storage media (disk, etc.).

Field Loadable Software

Synonym for “Onboard Loadable Software.” Per RTCA DO-178B, defines Field-loadable software as executable code or data tables that can be loaded without removing the system or equipment from its installation. Note: DO-178B does not draw a distinction between Field Loadable Software that is configured as part of the target hardware and Field Loadable Software that is configured as part of the airplane (i.e., LSAPs).

File Name

A "File Name" is the name of the file, without any information relative to its path – File names should include all extensions and delimiters (e.g., "filename.ext"). File names may contain uppercase and lowercase letters. File names on ARINC 665 media and references to them from within ARINC 665 files should be treated as if they were case-sensitive.

Hardware (HW)

Physical equipment, as opposed to computer programs, procedures, rules, and associated documentation. Contrast with software, firmware.

Header File

A specific file that contains information about the load that is needed to support the load process and software handling processes. Each load has one header file. See Section 2.2.3.1 for content and format.

Incompatibility Check

A determination if there are any known incompatibilities between two entities (e.g., software - target hardware, software - aircraft).

COMMENTARY

The lack of any known incompatibilities implies that the entities are compatible in the current environment within the thoroughness of the tests performed. However, testing for known potential incompatibilities cannot guarantee that the entities are totally compatible and/or interchangeable in every installation/usage. In many cases, factors that affect compatibility are not available to the function performing the incompatibility check.

Interchangeability

That quality which allows a component part to be substituted for another component part without affecting form, fit, function, or interchangeability of the parent component or system. Note: Being interchangeable does not imply that either part is certified for operation in any specific installation.

Interface

A shared boundary. An interface may be a hardware component to link two devices, or it may be a portion of storage or registers accessed by two or more computer programs.

Line Replaceable Unit (LRU)

A component which is designed to be removed and replaced by line maintenance personnel.

List-of Loads File

A specific file which contains the media set PN, media sequence number, and a list of the loads (and information about each load) which are on a specific media set.

Load (noun)

Synonym for “Loadable Software” and “Software Load.”

APPENDIX H
LOADABLE SOFTWARE TERMINOLOGY

Load (verb)

The process of transferring data into the program-memory of the "target hardware," also known as "dataload."

Load PN (Load Part Number)

The PN of the "Loadable Software Part" (not the PN of media set on which the software load is located).

Loadable Software

A software data set (i.e., group of files) designed for transferring into its "target hardware" without physically altering the hardware.

Loadable Software Airplane (Aircraft) Part (LSAP)

"Software" that is: (1) intended for transfer into its "target hardware" without physically altering the hardware or otherwise triggering the need for return-to-service conformity testing of the "target hardware," and (2) needs to be formally referenced independently from any other part (hardware or software) by airline or aircraft manufacturer's processes, and (3) is not configuration controlled as a component part of the target hardware, and (4) is configuration controlled as a component part of the aircraft.

COMMENTARY

Inherent in the definition of a LSAP is the concept that the LSAP is an independent, autonomous aircraft part from the target hardware. Installing a LSAP on the aircraft must not impact the conformity of the target or any other aircraft hardware. However, it may impact the aircraft conformity.

Loadable Software Airplane Parts (LSAPs) are a subset of the LSP class of parts. All provisions for LSPs in this document also apply to LSAPs.

Loadable Software Part (LSP)

"Software" that is intended for transfer into its "target hardware" without physically altering the hardware; and needs to be formally referenced independently from any other part (hardware or software) by airline or aircraft manufacturer's processes.

Loadsite

The position, place or memory location in the "target hardware" designed to contain a "load."

Load Source

The source of the data and header files that are being loaded (Mass Storage Device, CD-ROM, Gatelink, PC-Card, etc.).

An identifier for specific type of load classifies the LSP to general functional operations. The type is selected by supplier to correspond with the content of the Load Type Description field.

APPENDIX H
LOADABLE SOFTWARE TERMINOLOGY

Load Type

An identifier for specific type of load classifies the LSP to general functional operations. The type is selected by supplier to correspond with the content of the Load Type Description field.

Mass Storage Device (MSD)

A large capacity nonvolatile storage medium for software or data entities. Example: A hard disk drive or CD-ROM, which contains multiple files, loads, data bases, etc.

Media

Devices or material which act as a means of transferal or storage of software, for example; programmable read-only memory, magnetic tapes or disks, etc.

Navigation Data Base

A read-only data base of navigational information for upload to the flight management computer.

Non-Operational

Not performing its intended normal mission function. A unit may be "non-operational" when it is: failed, in software load mode, performing boot operations, aligning itself, etc.

NUL

ASCII no data character (value *0x0000*).

Onboard Load

Transfer of "loadable software" into "target hardware" while the hardware is installed on the aircraft.

Onboard Loadable Software

Synonym for "Field-Loadable Software."

Operational

Able to or performing its intended normal mission function.

Operational Program Configuration (OPC)

A load which contains information to control/select the flow/functionality of the OPS. This load replaces (or supplements) hardware program pin functions and may contain OPS option selections, installed equipment complement, aircraft structural configuration, engine type or other information that the OPS needs to know to properly operate in the specific environment. OPCs are generally very small and aircraft or customer specific.

The OPC is a classification of Option Selectable Software (OSS) LSPs.

Operational Program Software (OPS)

A load which contains application software for the "target hardware." OPSs are generally large, take longer to load and are fleet or model generic.

APPENDIX H
LOADABLE SOFTWARE TERMINOLOGY

Option Selectable Software (OSS)

Option Selectable Software is an LSP that the operator can modify within some boundaries without LRU re-certification. The system design should protect against inadvertent selections involving unsafe configurations for the Target HW. (Reference ARINC 667, RTCA DO-178).

Parallel Load

Parallel loading allows multiple target hardware of the same type to be simultaneously loaded with the same SW.

Part Number

A set of numbers, letters or other characters used to identify a configuration item.

Part Root Directory

A directory in the root directory of a media member, which is the topmost directory level for all files within a single load part number. This same directory name should be used on all media members which contain files for a given load part number.

Pathname

The File Pathname is the complete path to the file, without the name of the file. A Pathname should always begin at the root directory of the media member (indicated by a leading backslash). A Pathname should always finish with a backslash - When a Pathname includes one or more directory names, the Pathname is constructed with the most significant (i.e., parent) directory name first, followed by lower level (i.e., child) directory name(s). The backslash character (“\”) is used as the delimiter between concatenated directories.

Pre-Load (Preload)

The “shop load” of a “Loadable Software Airplane Part” into the same hardware it would reside in if the software were installed on the aircraft.

Note: Installation of a pre-loaded LRU on the aircraft does not conform the aircraft to its authorized software drawing configuration. It takes an independent aircraft software configuration verification (after LRU installation) to conform the aircraft to its authorized software configuration.

Pre-production Part

A pre-production part or system is used for development testing and is not intended for delivery.

Process

A collection of ordered activities performed to produce a definable output or product.

Production Part

A production-configured hardware or software part intended for delivery.

Program Memory

The nonvolatile memory that the load is intended to remain in when the target hardware is not in software load mode. Program memory does not include any buffer memory that data may reside in during data transfer.

APPENDIX H
LOADABLE SOFTWARE TERMINOLOGY

Protocol

A formalized set of rules by which computers communicate.

Root Directory

The directory level for any given media, at which the media file system is intended to be mounted.

Simultaneous Load

Simultaneous Load is independently loading multiple target hardware (which may be of different type) with software (which may be different) at the same. This basically requires 2 independent loader functions even though they may be both using the same interface bus and source media.

Shop Load (Bench Load)

Transfer of “loadable software” into “target hardware,” while the hardware is not installed on the aircraft.

Short Load

The concept of Short-Load is that the target hardware may only need to transfer (from the load source) a selected subset of the complete load in order to bring its program memory from the current bit image to the correct bit image for subject software PN. Short-Load is only valid if the process ensures (to the appropriate integrity) that the resulting target hardware program memory bit image is exactly the same as it would be if the complete software load were transferred.

Software

Data or code (executable or not) that defines, controls, or is used by its “target hardware” to perform its function.

Software Load

Synonym for “loadable software.”

Software Load PN

Synonym for “load part number.”

Software Loading (SW Loading)

Process of uploading software (including data) to the “target hardware.”

Software Part Number

Synonym for “load part number.”

Support File

Data associated with the LSP, such as description, Readme.txt, or Copyright statement, may be included in the content of the part, at supplier’s discretion. Some system types may anticipate specific file in form of a support file, as directed in selected sections of this document. Caution is given for LSP developers to minimize support file inclusion, recognizing any correction to a support file included within the definition of a load constitutes a distinct LSP.

APPENDIX H
LOADABLE SOFTWARE TERMINOLOGY

System

A group of components united by interaction or interdependence, performing various tasks but functioning as an integrated whole.

Target Hardware (Target HW)

The subject hardware of an operation. For example: the destination of the load, the hardware/LRU/location selected by the maintenance person as the destination of the load, the hardware the software is designed to operate in, etc.

Target HW ID (THW_ID)

Target HW ID identifies a type of loadable target hardware.

Target HW ID POS (THW_ID_POS)

The THW_ID_POS identifies a specific instance of loadable target hardware.

Upload (Up Load)

A data transfer from the software media (disks, etc.) to the “target hardware.”

Virus

A piece of software that installs itself on a computer system and reproduces without the user’s knowledge, and which may have a damaging effect on the computer system.

APPENDIX I REFERENCE GUIDE

This Reference Guide lists the references in **ARINC Report 665:** *Loadable Software Standards*. The references are categorized by their importance to the Portable Data Loader (PDL) developer, Airborne Data Loader (ADL) developer, and Target HardWare (THW) developer. The following numbers identify the categories:

1. Reference document required to implement the recommendations of ARINC Report 665.
2. Reference document with information that supports ARINC Report 615A.
3. Reference document that provides additional information.

ARINC 665 - Loadable Software Standards	PDL	ADL	THW
ANSI X9.30 (part 2) SHA-1 Hash Algorithm-1	1	1	1
ARINC Specification 429: *Mark 33 Digital Information Transfer System (DITS), Part 1, Functional Description, Electrical Interface, Label Assignments and Word Formats* [Equipment ID for THW ID only]	1	1	1
ARINC Report 615-3: *Airborne Computer High Speed Data Loader*	3	3	3
ARINC Report 615A: *Software Data Loader Using Ethernet Interfaces*	1	1	1
ARINC Specification 629: *Multi-Transmitter Data Bus*	3	3	3
ARINC Report 641: ***Logical Software Part Packaging for Transport***	**3**	**3**	**3**
ARINC Report 667: ***Guidance for the Management of Field Loadable Software***	**3**	**3**	**3**
ARINC Report 827: ***Electronic Distribution of Software by Crate (EDS Crate)***	**3**	**3**	**3**
ARINC Report 835: ***Guidance for Security of Loadable Software Parts Using Digital Signatures***	**3**	**3**	**3**
ARINC Specification 838: ***Loadable Software Part Definition Format***	**2**	**2**	**2**
ASCII - American Standard Code Information Interchange	2	2	2
ATA 2000 - Air Transport Association **(Airlines for America (A4A))**	2	2	2
ATA 2000 Bar Code Standard - Code 39	1	1	2
CAGE Code - Commercial And Government Entity Code	3	3	3
D6-55562-5 Header File	3	3	3
European Aviation Safety Agency (EASA) Regulatory Requirements	3	1	1
FAR 45.15 - replacement or mod part marking	3	3	3
Federal Aviation Authority (FAA) Regulatory Requirements	3	1	1
IETF RFC1321-The MDS Message Digest Algorithm	1	1	1
ISBN 1-57231-344-7 Windows 95, NT File Systems - Long File Names	2	2	2
ISO 9660 – CD Formatting (Joliet file specifications)	1	1	3
ISO/IEC 9529-1 “International Standard - Dimensions, Physical and Magnetic Characteristics” Section 7.1.	1	1	3
PC Card Standard	2	2	3
PSS ID Number Q140418 Fat Boot Sectors (Microsoft documentation)	2	2	2
RTCA DO-178 Software Considerations in Airborne Systems and Equipment Certification	3	1	1

APPENDIX J ~~FORM:~~ AIRPLANE LOADABLE SOFTWARE – REQUEST FOR MANUFACTURER'S CODE DESIGNATION

Airlines and organizations developing software should utilize a unique Manufacturer's Code designation. To request an MMM code, the preferred method is via a web application, found here:

http://www.aviation-ia.com/cf/manucode_form.cfm

Alternately, an MMM code can be requested via email using the form on the following page. Return the form via email to *manucode@sae-itc.org* or by fax to +1 301 383-1231.

APPENDIX J
AIRPLANE LOADABLE SOFTWARE – REQUEST FOR MANUFACTURER'S CODE DESIGNATION

Request for ARINC Report 665 Manufacturer's Code (MMM) Designation

From: ____

Fax: ____

Telephone: ____

Email Address: ____________________________________

Subject: Airplane Loadable Software - Request for Manufacturer's Code Designator

To: ARINC Industry Activities Staff
Manufacturer's Code Administrator
Fax: **+1 301 383-1231**
Email: *manucode@sae-itc.org*

REF: Manufacturer's Code Date: ________________

In accordance with ARINC Report 665, this is a request for assignment of a Manufacturer's Code, also known as a "MMM" Code.

The following information is required to process the Manufacturer's Code assignment.

Complete Customer Name: (Upper and Lower Case Required, 2 lines of 50) (Line 1: Company Name, Line 2: Configuration Management Group supplying software)

Corporate Address:

PREFERRED NEW CODE: _____________________
(Note: If your preferred new code is already assigned, we will call you and negotiate for an alternate.)

Phone:______________________ Fax: __________________________

Response:

ASSIGNED CODE: _____________________

ARINC Industry Activities Staff
Manufacturer's Code Administrator

APPENDIX K CALCULATING LOADABLE SOFTWARE PART CRC

K-0 Overview

This appendix offers orientation to, instructions for and examples of Loadable Software Cyclic Redundancy Codes (CRC). The material is presented in the sequence of:

- A conceptual CRC Algorithm
- A manual approach to calculating an 8-bit CRC
- A functional flow for automatic generation of a CRC
  - This flow incorporates a table build and look-up option to enhance process efficiency
- A sample **C** code program for calculating Loadable Software CRCs
  - The sample represents a 32-bit CRC
  - The code is designed in accordance with the functional flow provided above
  - Alternative size CRC generator code can be derived by appropriate alterations of variables
  - Modifiable variables are consistent with instructions given in Section 4.3
- Sample files and corresponding CRCs

These materials are given to assist in coordinating CRCs between software supplier and receiver, to assist in tracking violations of airplane software integrity. The code and instructions are given strictly as guidance in aligning processes and tools with Airplane Loadable Software CRC specifications. Sender and Receiver have responsibility for the CRC tools they elect to employ in their processes.

It is significant to recognize, there are multiple approaches to CRCs. Software may employ a number of CRCs each for a distinct purpose. The CRCs of this standard, address the packaging of software parts. Other CRCs may be applied internally, within the execution of the software, to insure processing integrity. No relationship between the packaging CRC and internal, processing CRCs should be established.

K-1Conceptual CRC Algorithm

The following is a conceptual CRC algorithm based on the formal definition. It cannot be implemented as specified, mainly because there is no computer hardware register R big enough to accommodate shifting an entire data block left by one bit. This algorithm is offered only as an aid to understanding.

APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

Definitions:

- A block B of binary data to be CRC checked is an ordered sequence of L bits indexed from (L-1) down to 0.

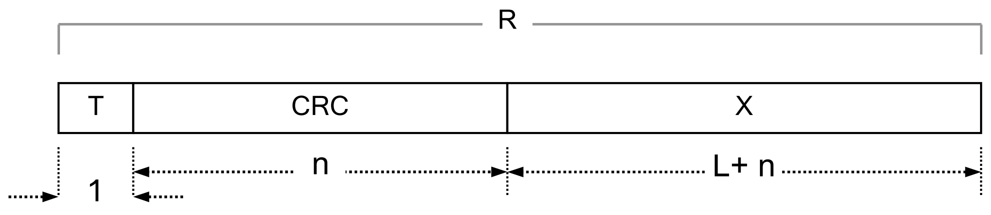
$$B = \{b_{L-1}, b_{L-2}, \dots, b_1, b_0\}.$$

- A CRC "generator" G is an ordered sequence of (n+1) bits indexed from n down to 0 such that $g_n = 1$ and $g_0 = 1$.

$$G = \{g_n, g_{n-1}, \dots, g_1, g_0\} = \{1, g_{n-1}, \dots, g_1, 1\}.$$

Imagine a hypothetical (and very long) bit shift register R whose entire contents can be shifted left one bit at each step of the computation; that is, its Most Significant Bit gets shifted off its left end, all of its other bits shift left one position, and a zero gets shifted into its Least Significant (or right-most) Bit position. Assume that R is partitioned into three contiguous sections that can be accessed individually by name:

- A high order bit T used for testing,
- n contiguous bits collectively called CRC, and
- (L+n) contiguous bits collectively called X.
- Note: in the algorithm given below the notation [T,CRC] means the concatenation of T and CRC



The algorithm

```
Set bit T to zero
Set each bit of CRC to zero
Initialize the L high order bits of X with B
Fill the remaining low-order bits of X with "n" zero bits
loop for each bit of X   (L + n times)
            Perform a one bit left logical shift on the entire register R
            if      T=1
                    Set [T,CRC] to [T,CRC] exclusive-OR G
            end if
end loop
```

The CRC section of R contains the result of the computation.

T	CRC	X
0	0 0 0 … 0 0 0	b_{L-1} b_{L-2} … b_1 b_0 0 0 0 … 0 0 0

APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

K-2Manual Approach for CRC Generation

A manual approach for calculating an 8 bit CRC is provided in Appendix E.

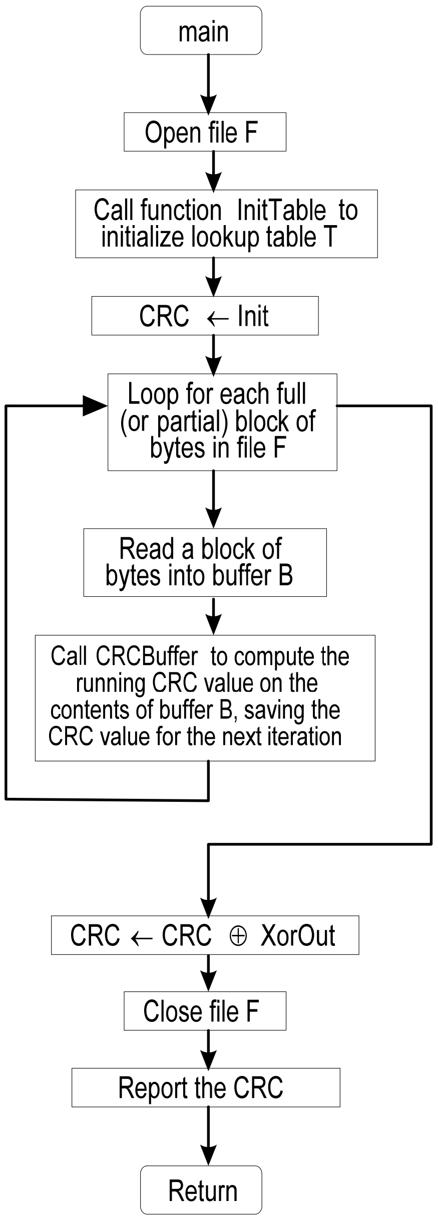
K-3A Functional Flow for Automatic Generation of a CRC

This section describes a table-driven algorithm for computing a cyclic redundancy code of width = n bits. Function main calls both InitTable and CRCBuffer.

Function main

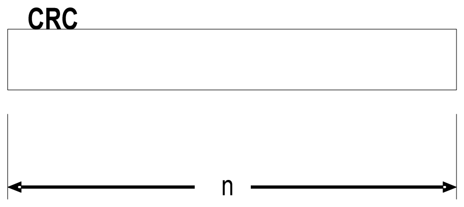
Computes an n-bit cyclic redundancy code CRC on a binary file F of data.

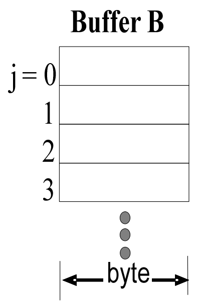
APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

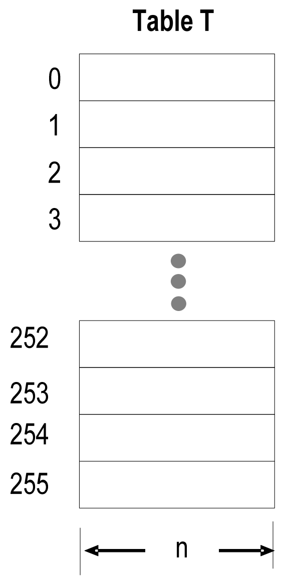

main
Open file F
Call function InitTable to initialize lookup table T
CRC ← Init
Loop for each full (or partial) block of bytes in file F
Read a block of bytes into buffer B
Call CRCBuffer to compute the running CRC value on the contents of buffer B, saving the CRC value for the next iteration
CRC ← CRC ⊕ XorOut
Close file F
Report the CRC
Return

Notation

B	Buffer to hold the current block of bytes
CRC	An n-bit cyclic redundancy code
F	A binary file of 8-bit bytes
Init	An n-bit constant used to initialize CRC
XorOut	An n-bit Exclusive-OR mask to be applied to the final CRC value
T	A lookup table with 256 entries, each n bits wide
⊕	Bit-wise Exclusive-OR operator
x ← a	Assign a to x







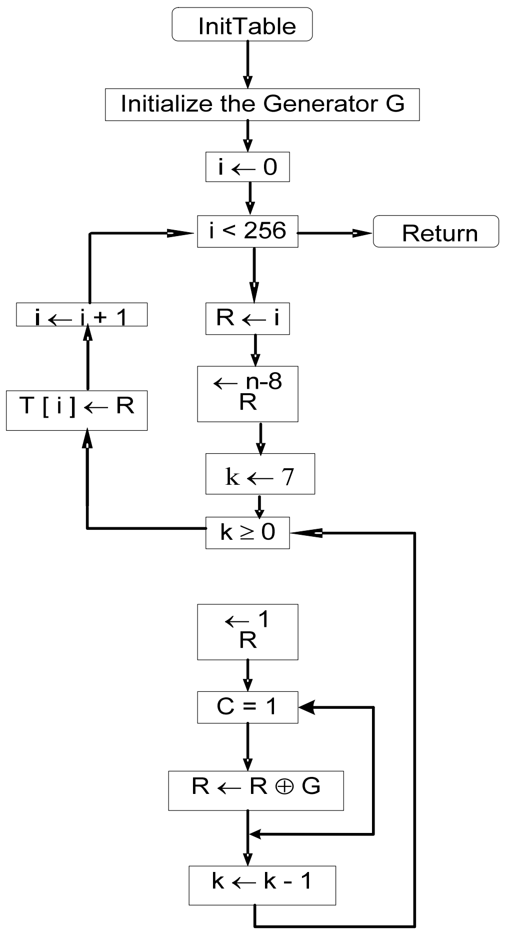
APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

K-3.1 Function InitTable

Initializes the 256 element lookup table T used to calculate the CRC.

Notation

C	Carry bit
G	Generator polynomial without its most significant bit (n bits wide)
i	Index into lookup table T
k	Bit index for bits in most significant byte of R: 7..0
n	Width of CRC in bits
R	Left shift register n bits wide. Last bit to be left shifted out of R is saved in C
T	Lookup table with 256 entries, each n bits wide
$\overset{\leftarrow m}{R}$	Left shift R by m bits (0s shifted into right end)
⊕	Bit-wise Exclusive-OR
x ←a	Assign a to x



Shift Register R Showing C

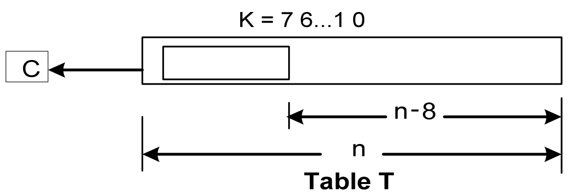
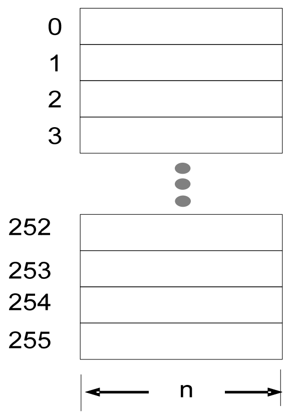


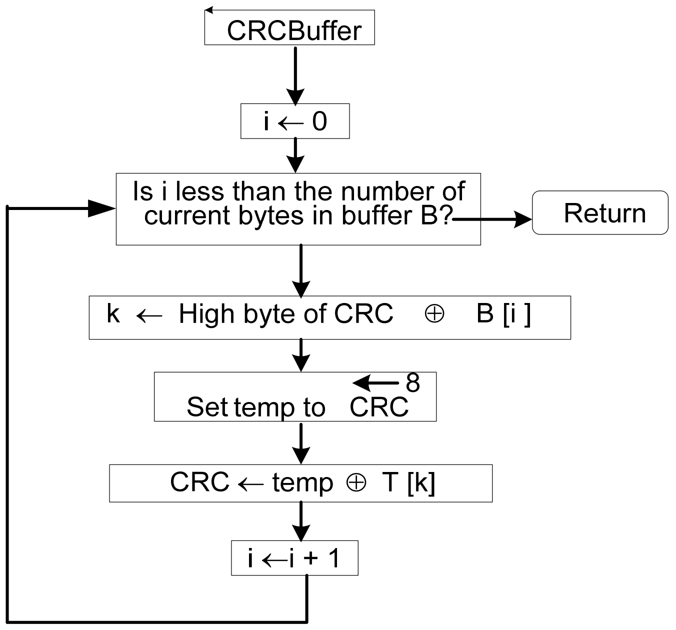
Table T



APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

K-3.2 Function CRCBuffer

Updates the running CRC computation per the contents of buffer B.


CRCBuffer
$i \leftarrow 0$
Is i less than the number of current bytes in buffer B?
Return
$k \leftarrow$ High byte of CRC $\oplus$ B [i]
Set temp to $\overset{\leftarrow 8}{\text{CRC}}$
CRC $\leftarrow$ temp $\oplus$ T [k]
$i \leftarrow i + 1$

Notation

B	Buffer to hold the current block of bytes
CRC	An n-bit cyclic redundancy code
i	Index into buffer B
k	Index into lookup table T
T	A lookup table with 256 entries, each n bits wide
$\overset{\leftarrow m}{\text{CRC}}$	Left shift CRC by m bits (0s shifted into right end)
$\oplus$	Bit-wise Exclusive-OR
$x \leftarrow a$	Assign a to x

CRC

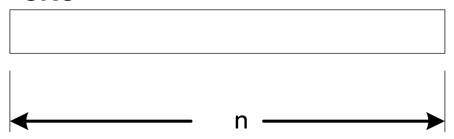
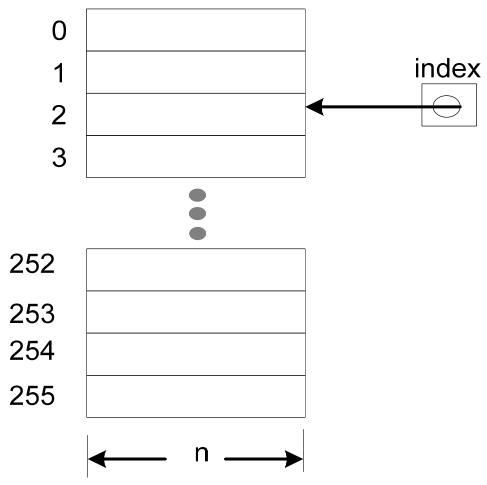
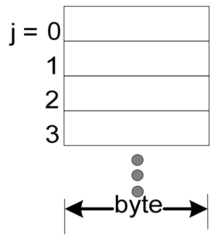


Table T



Buffer B



APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

K-4 Sample *C* Code Program for Calculating Loadable Software CRC

The following C code example represents logical structure of a software program designed in accordance with the CRC calculation methods depicted in Sections K-1 and K-3. The code is offered "AS IS" for instructive purposes only and may not produce consistent CRC values in variable environments and operating platforms. Creators of a CRC Generator based on this C code example assume full responsibility for the results of such generators.

The following example C code computes the ARINC 32-Bit Standard CRC on a specified file.

```
/*--------------------------------+-----------------------------------*\
|                    C R C - 3 2   E x a m p l e                       |
+----------------------------------------------------------------------+
| This program computes a 32 bit Cyclic Redundancy Code (CRC) on the   |
| contents of a single file = FILENAME.  The algorithm has the         |
| following specification:                                             |
|                                                                      |
|   Name:                           CRC-32                             |
|   Width:                          32 bits                            |
|                                                                      |
|   Polynomial:                     X**32 + X**26 + X**23 + X**22 +    |
|                                   X**16 + X**12 + X**11 + X**10 +    |
|                                   X**8  + X**7  + X**5  + X**4  +    |
|                                   X**2  + X     + 1                  |
|                                                                      |
|   Generator                       0x04C11DB7                         |
|   Initialization Value for CRC:   0xFFFFFFFF                         |
|   Input Bytes Reflected:          False                              |
|   Final CRC Reflected:            False                              |
|   XOR With Final CRC Value:       0xFFFFFFFF                         |
|                                                                      |
|   Average Probability of                                             |
|      Not Detecting an Error:      2.33 * [10**(-10)]                 |
|                                                                      |
|   Correct CRC for 256 Byte File                                      |
|      0x00 0x01 0x02...0xFE 0xFF:  0xB6B5EE95                         |
|                                                                      |
| Assumptions that could affect portability:                           |
|   char is 8 bits wide                                                |
|   unsigned long int is 32 bits wide                                  |
|                                                                      |
| Invocation:                                                          |
|   crc-32  FILENAME                                                   |
\*--------------------------------------------------------------------*/
// Include Files
     #include <stdlib.h>
     #include <stdio.h>

// Preprocessor Constants
     #define byte     char
     #define word_32  unsigned long int
```

APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

```
     #define BUFSIZE  1024        // Size of file buffer in bytes
     #define G        0x04C11DB7  // The generator G
     #define Init     0xFFFFFFFF  // Initialization value for CRC_value
     #define TABLEN   256         // Length of look-up table
     #define XorOut   0xFFFFFFFF  // To be XORed to final CRC_value

// Global (file scope) Variables
     static  byte     Buffer[BUFSIZE]; // The file buffer
     static  word_32  CRC_value;       // Holds the running CRC value
     static  FILE*    fp;              // File pointer
     static  size_t   nb;              // Number of bytes read from file
     static  word_32  table[TABLEN];   // Look-up Table

// Function Prototypes to Resolve Forward Referencing (See below)
     static void      CRCBuffer  (int);
     static void      InitTable  (void);

/*-----------------------------+---------------------------------*\
|                              m a i n                              |
\*--------------------------------------------------------------*/
void main(int argc, char* argv[])          // argv[1] is FILENAME
{
  // The operator must supply a FILENAME
     if (argc != 2)
     {
          printf("Error: Command line must contain a FILENAME\n");
     }

  // We must be able to open that file
     else if ((fp=fopen(argv[1],"rb"))== NULL)
     {
          printf("Error: Can't open input file\n");
     }

  // Otherwise compute CRC
     else
     {
       // Initialize the look-up table
          InitTable();

       // Initialize the CRC value
          CRC_value = Init;

       // Loop for each BUFSIZE (or less) block of bytes in FILENAME
          while (!feof(fp))
          {
            // Attempt to read a block of BUFSIZE bytes
               nb=fread(Buffer,sizeof(char),BUFSIZE,fp);

            // If any bytes were read, compute the running CRC_value
            // for them
               if (nb>0)
               {
                    CRCBuffer(nb);
               }
          }

       // Apply XorOut

          CRC_value ^= XorOut;

       // Close FILENAME
          fclose(fp);
```

APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

```
        // Report
           printf("CRC = %08X\n",CRC_value);
      }
}
```

APPENDIX K
CALCULATING LOADABLE SOFTWARE PART CRC

```
/*--------------------------------+-----------------------------------*\
|                          I n i t T a b l e                            |
|                                                                       |
| Initializes the look-up table                                         |
\*---------------------------------------------------------------------*/
static void InitTable(void)
{
     word_32 generator = G;        // CRC generator
     word_32 shift_reg;            // A shift register
     word_32 leading_bit;          // MSB of shift_reg before shift
     int     i;                    // Index into table 0..TABLEN
     int     k;                    // Bit index into byte 7..0

     for (i=0; i<TABLEN; i++)
     {
          shift_reg = ((word_32)i << 24);
          for (k=7; k>=0; k--)
          {
               leading_bit = shift_reg & 0x80000000;
               shift_reg = shift_reg << 1;
               if (leading_bit)
               {
                    shift_reg = shift_reg ^ generator;
               }
          }
          table[i] = shift_reg;
     }
}

/*--------------------------------+-----------------------------------*\
|                          C R C B u f f e r                            |
|                                                                       |
| Computes the running CRC_value for the current Buffer                 |
\*---------------------------------------------------------------------*/
static void CRCBuffer(int nb)
{
     int i;   // Byte index into buffer
     int k;   // Index into look-up table

  // Loop for each byte in Buffer
     for (i=0; i<nb; i++)
     {
       // Compute index into look-up table for the current byte
          k = ( (CRC_value>>24) ^ (int)Buffer[i] ) & 0xFF;

       // Update the running CRC_value for the curtrent byte
          CRC_value = (CRC_value << 8) ^ table[k];
     }
}
```

APPENDIX L CRC STANDARD REFERENCE FILES FOR SOFTWARE DATA LOADING

L-0 Introduction

This appendix contains a list of test files for checking CRC software algorithms. These files can be used by developers of software data loaders to test the CRC calculating functionality of their data loader. These files will enable developers to ensure that their data loaders conform to ARINC 665 CRC calculation standards.

The description of these files and CRCs are provided in this appendix. The test data comprising these files are posted on the ARINC IA Website in computer data formats. See the Software Data Loader Subcommittee webpage, here: *http://www.aviation-ia.com/aeec/projects/sdl/index.html*.

Software data loader developers and users may download these files to check the implementations of their CRC software algorithms.

L-1 Standard CRC Reference File Descriptions

Table L-1 provides the file name, size, and content of the standard CRC Reference Files.

Table L-1 – Standard CRC Reference File Descriptions

File Name	Size (in bytes)	Content
CRC_T01A.rom	0	Empty
CRC_T02A.rom	128	All 0xFF
CRC_T03A.rom	100	All 0x00
CRC_T04A.rom	256	128 x (0xAA55)
CRC_T05A.rom	3976	Random values
CRC_T06A.rom	18152	Random values
CRC_T07A.rom	34816	Random values
CRC_T08A.rom	34817	One byte more than CRC_T07
CRC_T09A.rom	1758480	Random values
CRC_T10A.rom	61	Ethernet frame
CRC_T11A.rom	256	Values of 0x00 through 0xFF
CRC_T12A.rom	11	String "ACM12345678"
CRC_T13A.rom	15	String "ABCDEFGHIJKLMNO"

L-2 CRC Values for Reference Files

This section provides the file names and CRC values for reference files for 8-, 16-, and 32-bit CRCs.

L-2.1 CRC 8-Bit Test Results

Table L-2.1 - Test File Results for 8-Bit CRC's

File	8 Bit CRC
CRC_T01A.rom	0x00
CRC_T02A.rom	0x00
CRC_T03A.rom	0x00
CRC_T10A.rom	0xE8
CRC_T12A.rom	0x47
CRC_T13A.rom	0x40

L-2.2 CRC 16-Bit Test Results

Table L-2.2 - Test File Results for 16-Bit CRCs

File	16 Bit CRC
CRC_T01A.rom	0xFFFF
CRC_T02A.rom	0x1DA3
CRC_T03A.rom	0x4634
CRC_T04A.rom	0x1D7E
CRC_T05A.rom	0xA208
CRC_T06A.rom	0xA12C
CRC_T07A.rom	0x2DA3
CRC_T08A.rom	0xCF07
CRC_T09A.rom	0x9EB1
CRC_T10A.rom	0xD8D2
CRC_T11A.rom	0x3FBD

L-2.3 CRC 32 Bit Test Results

Table L-2.3 - Test File Results for 32 Bit CRC's

File	32 Bit CRC
CRC_T01A.rom	0x00000000
CRC_T02A.rom	0x322AB4A6
CRC_T03A.rom	0x53631199
CRC_T04A.rom	0xC2F270BC
CRC_T05A.rom	0x96142DCA
CRC_T06A.rom	0xAE34897C
CRC_T07A.rom	0x55A1228D
CRC_T08A.rom	0x3109EB62
CRC_T09A.rom	0x239B226B
CRC_T10A.rom	0xC0BB3B8E
CRC_T11A.rom	0xB6B5EE95

L-3 Disclaimer

The user of these sets of files and CRC results are responsible for the use and interpretations of the results as they pertain to the checking of their CRC software algorithms. ARINC and the AEEC accept no responsibility for the checking of the user's implementation of the CRC calculations and their interpretation within the user's application.

APPENDIX M CONSIDERATIONS FOR IMPLEMENTING SUPPLEMENT 2 TO ARINC REPORT 665

This appendix is specific to implementers who create or maintain LSPs specific to Supplement 2 to ARINC Report 665, published August 30, 2002. The material was originally published as a working paper as “Clarifications to ARINC 665-2.”

It is important to note that air transport software should be created, maintained, and dispositioned using the latest supplement of ARINC Report 665.

M-1 Introduction

This appendix provides clarifications to specific ARINC 665 issues that are judged to be of particular importance. When new Loadable Software Parts (LSPs) are created in accordance with ARINC 665-2, the preferred interpretation identified in this appendix should be implemented. This appendix is only concerned with issues that are known to cause incompatibility between tools or which cause dataloading failures that prevent LRUs from being uploaded. However, it is not the intention of this appendix to cause any existing software to be modified or any existing loadable software parts to be changed. The purpose of this appendix is to provide clarifications to the community and to provide an interpretation that should be used on any new loadable software parts built to ARINC 665-2.

The goal of this document is to clearly identify the issues, and provide a classification system for the variations that are in current use, so that users can:

- Understand the issues
- Classify their tools and LSPs
- Obtain tools that work for their LSPs
- Implement new software for target hardware using the preferred interpretations

Each issue identified in Section M-3 of this document is structured as follows:

- Problem Statement
- Preferred Interpretation
- History/Discussion
- Alternate LSP Implementations
- Alternate Tool Implementations

M-2 Background

M-2.1 Before ARINC 665

Prior to the release of ARINC 665, common loadable software practices were controlled by agreement between aircraft manufacturer and aircraft system supplier. These agreements offered a framework around which much of the original ARINC 665 standards were patterned. Additionally, ARINC 665 served to accommodate and promote emerging technologies and advanced designs. The standard was designed to accommodate primary objectives of:

- Retain and promote existing loadable software practices

- Enable software load over an Ethernet protocol
- Allow loadable software parts to be distinct and separable from host media
- Allow for growth in content and application of loadable software

M-2.2 ARINC 665

The objective of ARINC Report 665 is to define Loadable Software Airplane Parts in such a way as to offer compatibility and interoperability between airplane systems and tools handling loadable software. This is accomplished by establishment of "common principles and rules" addressing "part numbering, content, labeling and formatting."

Successive updates of ARINC 665 were designed to retain the fundamental structure of loadable software parts. Added data fields were strategically placed to maximize compatibility between format versions to avoid redesign of systems applying the parts, so long as the fields are consistently defined and applied.

The primary tools involved with ARINC 665 LSPs are:

- ARINC 615A dataloaders
- ARINC 615A target dataloading engines
- ARINC 665 LSP/~~media set~~ creation tools
- ARINC 665 LSP/~~media set~~ inspection or verification tools

Other tools may also be involved, for example:

- Target dataloading engines that accept ARINC 665 parts.
- Import tools used to move LSPs into a repository so that it may be used by some other tool (e.g., a dataloader). Import tools usually parse and check an LSP, and so are subject to ARINC 665 issues.
- Electronic distribution tools. Distribution tools may parse and check an LSP prior to distribution and/or after distribution.

In the preparation of ARINC 665, the attention given to provide concise and complete field descriptions proved successful with the exception of a few fields. Each update of ARINC 665 attempts to clarify and align design and process issues. For example, ARINC 665-3 included many clarifications, but failed to explain how these clarifications were to apply back to earlier part format versions.

Accordingly, Appendix M identifies key issues of ARINC 665-2 for which clarification and instruction have proven necessary. Designers of new Loadable Software Parts and systems are expected to incorporate these clarifications in their use of ARINC 665-2.

M-3 ARINC 665-2 Identified Clarifications

M-3.1 Unique Filename Convention

M-3.1.1 Problem Statement

The ARINC 615A load protocol allows the target hardware to request files from a number of LSPs during one load session. The software loader looks for the requested file(s) from all LSPs which are currently being loaded. It may send the wrong file if the file names are not unique among LSPs. The symptom most often seen is a target side CRC failure as a result of this file confusion.

This problem may occur regardless of whether the identical file names are located in the same or separate LSPs. This problem may also occur regardless of whether the identical file names are located in the same or separate MSPs.

In ARINC Report 665-2, Section 2.2.2, the requirement is stated:

“The filename should be assigned such that it is unique for all files defined by the Manufacturer’s Code.”

The intent of this requirement is to avoid file conflicts where the same file name is used for different file contents. It was originally intended that files would be renamed for each LSP using the part number as part of the file name. Where the same file name is used for the same contents, this issue does not generally cause a problem. Increasing use of COTS on the airplane complicates this design.

M-3.1.2 History/Discussion

The ARINC 665-2 requirement:

“The filename should be assigned such that it is unique for all files defined by the Manufacturer’s Code”

is recognized for its intent to avoid file conflicts. However, when interpreted in isolation, this requirement implies unreasonable and unnecessary obligations such as preventing the reuse of static reference or library files common between LSPs and renaming of retained files in subsequent release of an LSP. To allow for reuse and to avoid unnecessary burden, the requirement statement was corrected in ARINC 665-3 to read:

“The filename should be assigned such that it is unique for each load associated to the Manufacturer’s Code.”

This guidance does not invalidate previously released LSPs but accommodates, acknowledges, and is aligned with existing practices.

This ARINC 665-3 guidance has implications on build and verification tools which anticipate or enforce the rigidity of the ARINC 665-2 requirement. Static reference or library files common between LSPs in a given Media Set Part (MSP) may now be repeated on that MSP.

This guidance makes allowance for an associated condition where a file name may be repeated in distinct LSPs on the MSP, yet reflect altered content. Instructions for distinguishing instances of repeated files are offered in Section 3.2.4.1 of ARINC Report 665-3.

M-3.2 LSP Implementations

M-3.2.1 Implementation A (Preferred)

Following the original intent of ARINC 665-2, all file names for a given MMM Code are unique because they reflect the MMM Code, LSP part number, etc.

COMMENTARY

Implementation A implies that every LSP part number roll requires all LSP file names to be changed regardless of whether or not the contents have changed. File Part Numbers may or may not change. This imposes a burden on the part creation process but it completely eliminates any duplicate file name issues from arising.

M-3.2.2 Implementation B (Acceptable)

The file names of all LSPs which can be loaded to a particular target hardware are unique. File names do not conform to the standard in that MMM Code and part number are not necessarily included in the name of each LSP file. This typically is used where a file naming convention is applied in common across a set of LSPs that may come from different manufacturers but the naming convention prevents duplicate file name issues.

M-3.2.3 Implementation C (Discouraged)

Common file names may be used for different file contents across multiple LSP, but all file names are unique within an LSP. If different LSPs have common file names and the ARINC 615A data loader is directed to load multiple LSPs in a single upload operation, this issue may cause problems.

COMMENTARY

It should be noted that future ARINC 665-3 (Format 8004) LSPs may have common file names for different file contents, if the ARINC 615A-3 Part Number and Checksum TFTP options are used, they will permit these common file names to be loaded correctly.

The filename is no longer sufficient to identify a unique file. Either Part number and filename or filename and CRC are required to identify a particular file. When an LRU requests files for a set of LSPs, the LRU must distinguish among the duplicate files using either the Part Number or Checksum TFTP options as defined in ARINC 615A-3.

It is the responsibility of all LSP creators to ensure that a part does not contain duplicate file names with differing contents and with identical CRCs. Since the data file CRCs are only 16-bit CRCs, which are only valid over a file that is no larger than 4 Kbytes, it is theoretically possible to violate this statement and still be valid with files larger than 4 Kbytes in size.

M-3.2.4 Implementation D (Strongly Discouraged)

Common file names used for different file contents within an LSP. It should be noted that ARINC 665-2 and ARINC 665-3 prohibit this usage.

COMMENTARY

It should be noted that ARINC 665-3 (Format 8004) LSPs may have common file names for different file contents; if the ARINC 615A-3 Checksum TFTP option is used, it will permit these common file names to be loaded correctly.

Implementation D uses have been seen in legacy systems.

ARINC 615A-2 data loaders will likely fail with Implementation D parts.

ARINC 615A-3 data loaders may or may not fail with Implementation D parts.

The filename is no longer sufficient to identify a unique file. Either part number and filename or filename and CRC are required to identify a particular file. When an LRU requests files for a set of LSPs, the LRU must distinguish among the duplicate files using either the part number or checksum TFTP options as defined in ARINC 615A-3.

It is the responsibility of all LSP creators to ensure that a part does not contain duplicate file names with differing contents with identical CRCs. Since the data file CRCs are only 16-bit CRCs which are only valid over a file that is no larger than 4 Kbytes, it is theoretically possible to violate this statement and still be valid with files larger than 4 Kbytes in size.

M-3.3 Guidance on Tool Implementations

When parsing a set of LSPs it may be possible to find duplicate file names which should contain identical data. In this case, it is acceptable to choose any file matching the given file name.

It is the responsibility of all LSP creators to ensure that a part does not contain duplicate file names with differing contents with identical CRCs.

The following table provides desired operational characteristics by LSP implementation and tool classification.

	Imp. A LSP	Imp. B LSP	Imp. C LSP	Imp. D LSP
Data Loader	All data loaders work properly	All data loaders work properly	Data loader may fail one of the multiple LSPs loaded	ARINC 615A-3 data loaders will work. Older data loaders will not work.
Target Hardware Engine	All engines work properly	All engines work properly	Engine may see a data file CRC error for some transfers	ARINC 615A-3 engines will work. Older engines will not work.
Creator	Only Imp. A LSPs are created.	Only Imp. B LSPs are created.	Creator does not take subsequent multiple LSP loads into account.	Creator has no effect.
Inspector	Errors given for non-Imp. A file names	Errors given for non-Imp. B file names	N/A	Should produce an error

M-3.4 Data File Length (DFL)

M-3.4.1 Problem Statement

Before the publication of Supplement 3 of ARINC 665 in 2005, ARINC 665 specified that the length of all data files should be an even number of bytes and no guidance was provided for files which are an odd number of bytes in length.

In addition, the field description in ARINC 665-2, Section 2.2.3.1.2.1 was inadvertently overlaid by a copy of the text in the section which preceded it.

As a consequence, manufacturers had no explicit guidance on how to treat a data file when its length was an odd number of 8-bit bytes and were missing a clear description of how to use the length field and interpreted as they saw fit. It should be noted that this does not apply to support files. Only data files are affected.

M-3.4.2 History/Discussion

The field description in ARINC 665-2 was inadvertently overlaid by a copy of the text in the section which preceded it. The ARINC 665-2 text is:

“The Data File PN field is an 8-bit ASCII character string whose length is defined by the Data File PN Length field. The field is allocated...”

The text does not pertain to DFL. The SDL Subcommittee addressed this quickly after the ARINC 665-2 standard was published and the meeting report described the correction. Here is the correction that was published (years later) in the ARINC 665-3 revision.

“The data file length is the number of 16-bit words in the data file. A half-word at the end of a data file should be counted as a complete word.”

The correction addressed the problem of the overlaid text. However, an unfortunate outcome was that some users were already interpreting the field to be the number of 8-bit bytes. Another unfortunate outcome of the correction is that it introduced another problem – the corrected text does not allow one to accurately convey the length of odd-length files.

The ARINC 615A-2 standard assumes that data files will be even length. However, there are many LSAPs that contain odd-length files. In actual practice, dataloaders and tools must accommodate odd-length files in order to load these LSAPs.

The SDL Subcommittee took another corrective action to address the odd-length file issue. A new field called Data File Length in Bytes (DFLB) was added to the ARINC 665-3 LUH file. The DFL field was left as a word count field for backwards compatibility, and the new DFLB field conveys the accurate length information needed for odd-length files.

The result is that the DFL field cannot be relied upon to give an accurate length unless: 1) you know that the field has a byte count or 2) you know that the DFL field has a word length, and you know that your data files are even length.

The issue is further complicated by the question of rounding up or down in the case where a word count is placed in the field and the file is odd length. ARINC 665-2 did not address this case. However, ARINC 665-3 document (see above) addressed this with a “rounding up” decision.

The support file length field does not have this issue. In ARINC 665-2, that field was specified as a byte count. Therefore, none of these problems happened with that field.

Known interpretations include:

- General implementation is consistent with the preferred design

- In some LSP implementations, the number of 16-bit words has been rounded-up or rounded-down when there are an odd number of bytes in the file.
- Some LSP implementations interpreted the data file length as a count of 8-bit bytes

M-3.4.3 LSP Implementations

For LSPs prepared subsequent to release of this bulletin, observe the specification found in the preferred implementation in Section 3.2.3.1, with the following accommodations:

- Data files where the final word is comprised of a single 8-bit byte, shall include the half word in its length count.
- Support tools shall not restrict parts of a last half word, where the count is off by one.
- Data files length count representing the number of 8-bit bytes shall be invalid.

Note: If the Data File Length field reflects a count of one less word, a warning of "Odd-byte Data File Length field round-down error" may be offered. If the remaining LSP header file content reconciles to data file CRC checks, the LSP should be approved. Intent of the warning is to simply expose odd-byte data files for processes sensitive to this condition.

M-3.4.3.1 Implementation A (Preferred)

The preferred interpretation as specified in ARINC 665-3 is:

The data file length is the number of 16-bit words in the data file and the data file should contain an even number of bytes.

In order to fully comply with the preferred implementation, the length of the actual data file must be an even number of bytes and thus it can be exactly described as a number of words.

COMMENTARY

The confusing wording found in ARINC 665-2 is replaced by clarifying Section 2.2.3.1.37 of ARINC 665-3.

M-3.4.3.2 Implementation B (Acceptable)

An odd byte-length file is rounded up to a 16-bit word count. This follows the ARINC 665-3 wording.

COMMENTARY

Other indicators of file length (file length as reported by a file system or transport mechanism, e.g., TFTP) should be used as the definitive file length and the contents of the header file as a guide.

M-3.4.3.3 Implementation C (Strongly Discouraged)

An odd byte-length file is rounded down to a 16-bit word count.

COMMENTARY

Other indicators of file length (file length as reported by a file system or transport mechanism, e.g., TFTP) should be used as the definitive file length and the contents of the header file as a guide.

M-3.4.3.4 Implementation D (Strongly Discouraged)

Any other number, e.g., insertion of a byte-length instead of a 16-bit word count, is inserted in this field. This is considered to invalidate a part as it does not conform to the standard and should be corrected.

M-3.4.4 Guidance on Tool Implementations

The following table provides desired operational characteristics by LSP implementation and tool classification.

	Imp. A LSP	Imp. B LSP	Imp. C LSP	Imp. D LSP
Data Loader (Note 1)	Don't care	Don't care	Don't care	Don't care
TH Engine	DFL may be used	Note 2	Note 2	Note 3
Creator	Creates Imp. A LSPs	Creates Imp. B LSPs	Creates Imp. C LSPs	Creates Imp. D LSPs
Inspector	Note 4	Note 4	Note 4	Note 4

Note 1: The loader should not care. It has access to the original file and while it may note the discrepancy between the header's indicated file length and the actual file length this does not prevent operation of the loader.

Note 2: The ARINC 615A data loading engine can use this information as an approximate file size. It can successfully load these type files because it uses an independent file length as obtained from TFTP, i.e., the actual length that was transmitted. This length is always accurate. It successfully loads both byte and word count DFL fields regardless of whether the DFL was rounded up or rounded down, in the case of an odd length file. There may be other ways of obtaining an accurate file size.

However, non-ARINC 615A data loading engines may not load these LSPs successfully without an independent file size determination mechanism.

Note 3: The data loading engine either knows that the field is a byte-length or it has an independent means of determining the file size.

Note 4: Inspectors are typically paired to the specific LSPs. They will give warnings or errors when the expected values are incorrect.

M-3.5 Zero Number of Support Files

M-3.5.1 Problem Statement

ARINC 665-2, Section 2.2.3.1.24 specified, for LSPs with no support files, to assign zero to the Pointer to Number of Support Files field. However, proposed wording to omit the field not pointed to (i.e., the Number of Support Files (NSF) field) failed to be included in the published ARINC 665-2.

M-3.5.2 Preferred Interpretation

The preferred interpretation is:

"If the Pointer to the Number of Support Files field is set to 0x0000 then the Number of Support Files field and subordinate Support File fields should be omitted from the file."

ARINC 665-3 corrected this error by including the above requirement for format 8003.

M-3.5.3 History/Discussion

In ARINC 665-2, Table 2.2.3.1 failed to recognize Number of Support Files may be a value of zero or more. Strict interpretation of ARINC 665-2 may have generated LSPs with a zero Pointer to Support Files field and a zero-filled Number of Support Files field.

Consequently, LSP support tools need to account for format 8003 parts which may or may not include a Number of Support Files field when the "Pointer to Support file list" contains a zero value (i.e., 0x0000).

M-3.5.4 LSP Implementations

M-3.5.4.1 Implementation A (Preferred)

The NSF field is not inserted if the Pointer to Support Files field is set to zero.

COMMENTARY

An LSP parser should be constructed such that it is indifferent to the presence or omission of the NSF field.

M-3.5.4.2 Implementation B (Acceptable)

The NSF field is inserted with a zero value if the Number of Support Files field is set to zero.

COMMENTARY

An LSP parser should be constructed such that it is indifferent to the presence or omission of the NSF field.

M-3.5.5 Guidance on Tool Implementations

The following table provides desired operational characteristics by LSP Implementation and tool classification.

	Imp. A LSP	Imp. B LSP
Data Loader	Don't care	Don't care
TH Engine	Note 1	Note 2
Creator	Creates Imp. A LSPs	Creates Imp. B LSPs
Inspector	Note 3	Note 4

Note 1: TH engines built to accept Implementation A LSPs will also accept Implementation B LSPs since they do not follow the pointer to the Number of Support Files field.

Note 2: TH engines built to accept Implementation B LSPs will fail when they encounter an Implementation A LSP because they will expect a zero-value for the Number of Support Files field.

Note 3: An Inspector that is built to verify Implementation B LSPs may give warnings when is inspects an Implementation A LSP.

Note 4: An Inspector that is built to verify Implementation A LSPs may give warnings when is inspects an Implementation B LSP.

M-3.6 Media CRC

M-3.6.1 Problem Statement

The media CRC was defined in ARINC 665-2, it was intended to be carried in the Media Volume Label field per Section 3.2.1. In ARINC 665-3, the salient objective is met by other integrity measures, i.e., the Load CRC and the Data File CRCs. The use of the Media CRC in the volume label is discouraged as the volume label is not always retrievable from different operating systems. Furthermore, the Media CRC is not stored anywhere in any media file. Therefore, use of the Media CRC is discouraged because it was not always available for verification of the media.

M-3.6.2 Preferred Interpretation

The media CRC should not be implemented, and if it is inserted in an LSP, it should be ignored.

M-3.6.3 History/Discussion

See Problem Statement above.

M-3.6.4 MSP Implementations

M-3.6.4.1 Implementation A (Preferred)

The media CRC is not calculated and not inserted into the MSP Media Volume Label.

M-3.6.4.2 Implementation B (Acceptable)

The media CRC is calculated and is inserted into the MSP Media Volume Label.

COMMENTARY

This may be a required by specific MSP recipients.

If the CRC is to be calculated, the MSP file set and order must be determined in accordance with ARINC 665-2, Section 4.3.4.

APPENDIX M
CONSIDERATIONS FOR IMPLEMENTING SUPPLEMENT 2 TO ARINC REPORT 665

M-3.6.5 Guidance on Tool Implementations

An MSP parser (e.g., an MSP validation tool) should be constructed in such a way that it is not affected by this issue.

M-4 Conclusions

Implementers of new Loadable Software Parts following ARINC Report 665-2, are expected to observe the Preferred Implementations stated in Appendix M.

Implementers of inspection tools for LSPs should make tools that identify and tolerate all Preferred or Acceptable Implementations and identify and report appropriate warnings and errors for the Discouraged or Strongly Discouraged implementations.

Implementers of data loaders that are intended for a wide range of LSPs should accept the Preferred or Acceptable Implementations above. However, there may be LSPs which have Discouraged or Strongly Discouraged implementations which may need to be accommodated.