WHITE PAPER

ON KNOWN ERRATA ON SPECIFICATION ARINC 653

(AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE)

Prepared by the Application/Executive (APEX) Subcommittee

VERSION: 1.0

PUBLISHED: September 2021

ARINC SPECIFICATION 653 ERRATA WHITE PAPER
TABLE OF CONTENTS

Version	Date	Effect on	Reasons for revision
1.0	September 2021	All	Initial release

Note: revision marks are used between two consecutive versions.

Version 1.0, September 2021

0.0 INTRODUCTION

0.1 Purpose

This White Paper contains description of the known errata against the **ARINC Specification 653:** *Avionics Application Software Standard Interface* standard (a.k.a. APEX) Parts 1, 2 and 4 from ARINC 653-1 (2003 edition) to their latest releases at the date of release of this White Paper. This includes errata that was present in their past releases, but has been resolved. Corrections to the applicable Parts will be made as part of the work done per APIM 21-007, concluding in October 2023. New versions of this White Paper may be released at the sole discretion of the ARINC 653 Sub-Committee.

0.2 ARINC 653 Overview

As detailed in its Part 0, the primary objective of ARINC 653 is to define a general purpose APplication/EXecutive (APEX) interface (API = Application Program Interface) between the Core Software (CSW) of an Avionics Computer Resource (ACR) and the application software.

ARINC 653 is organized in seven parts, available at *ARINC Standards Store*:

- Part 0 – Overview of ARINC 653
- Part 1 – Required Services
- Part 2 – Extended Services
- Part 3A – Conformity Test Specification for ARINC 653 Required Services
- Part 3B – Conformity Test Specification for ARINC 653 Extended Services
- Part 4 – Subset Services
- Part 5 – Core Software Recommended Capabilities

0.3 ARINC 653 Evolution – Latest Releases

The ARINC 653 document set is being developed in parallel. Refer to Part 0 for more details. The structure and content of the documents has evolved as shown in Table 1; Figure 1 provides, where relevant, compatibilities between these Parts:

Table 1 – ARINC 653 Document Evolution

ARINC 653	Part 0 Overview	Part 1 Required	Part 3A Test Part 1	Part 2 Extended	Part 3B Test Part 2	Part 4 Subset	Part 5 Core Cap
Latest Document	Supp 3 (2021)	Supp 5 (2019)	Supp 2 (2021)	Supp 4 (2019)	Supp 1 [TBD, on hold]	Base Part 4 (2012)	Supp 1 (2019)
Previous Supplement	Supp 2 (2019)						
Previous Supplement	Supp 1 (2015)	Supp 4 (2015)		Supp 3 (2015)			Base Part 5 (2014)
Previous Supplement	Base Part 0 (2013)	Supp 3 (2010)	Supp 1 (2019)	Supp 2 (2012)	Base Part 3B (2019)		
Previous Supplement		Supp 2 (2005)	Base Part 3A (2014)	Supp 1 (2008)			
Previous Supplement		Supp 1 (2003)	Base Part 3 (2006)	Base Part 2 (2007)			
Previous Supplement		Base Part 1 (1997)					

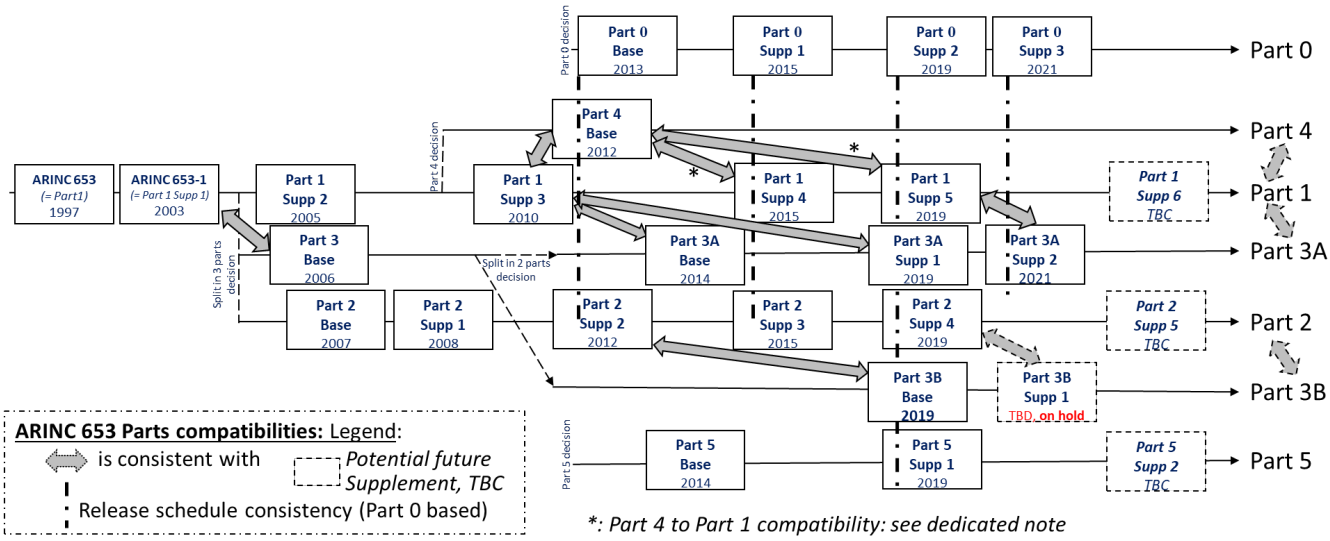


Figure 1 – ARINC 653 Parts compatibilities

Note: Part 4 to Part 1 compatibility: Although Part 4 was issued earlier in time, the Part 4 Base document is compatible with more recent Part 1 Supplements 4 and 5 (on top of Supplement 3 to which it explicitly refers);

However, the following restrictions apply:

- Multi-core specific aspects introduced in Supplements 4 and 5 are not supported,
- Changes to type definitions within Ada and C interface specifications need to be considered while porting an existing Part 4 compatible application to a Part 1 Supplement 4 or 5 compatible core module.

Note: TBD refers to documents that are currently being drafted. For further, exhaustive details about the changes introduced in a given supplement, refer to dedicated appendixes in the applicable supplement to each respective part.

0.4 Errata definition and identification

Errata are considered to be any inconsistency, ambiguity (e.g. text that could be interpreted in several ways), missing information, or typo in the standard that could be potentially detrimental to implementation. Consequently, they can only be relative to Parts 1, 2 or 4.

Errata are identified by ARINC 653 Part and location within that part; the problem statement includes the Supplement in which it first appeared. The resolution includes the Supplement in which the correction was first introduced, if the errata has been resolved; all impacted Supplements in between are also identified. For errata that is present in current releases, the proposed resolution or workaround is included.

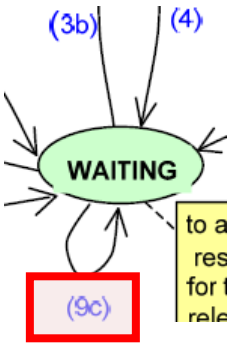
1.0 ARINC 653 PART 1 ERRATA

This section contains descriptions of the known errata in ARINC 653 Part 1.

1.1 Section 2.3.2.2.1.2, Figure 2.3- Process States transitions: Waiting-Waiting transition

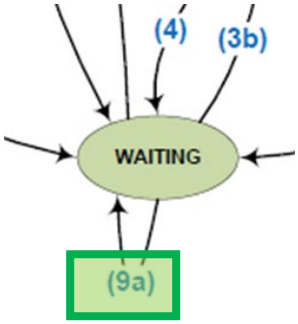
Problem Statement:

In Supplement 3, the right side Waiting-Waiting transition refers to “(9c)”, not “(9a)”:



Resolution:

In Supplement 4, the right side Waiting-Waiting transition now refers to “(9a)”:



Supplement Impacted:

Only Supplement 3 is impacted.

1.2 Section 3.3.2.13 – UNLOCK_PREEMPTION: LOCK_LEVEL when NO_ACTION

Problem Statement:

In Supplement 1, UNLOCK_PREEMPTION does not set the value of LOCK_LEVEL when the RETURN_CODE is NO_ACTION.

Resolution:

In Supplement 4, LOCK_LEVEL is now set to the current value of the partition's lock level when NO_ACTION is returned:

```
if (the current process is the error handler process or OPERATING_MODE is not
NORMAL or no process has preemption locked) then
   LOCK_LEVEL = current value of the partition's lock level;
   RETURN_CODE := NO_ACTION;
```

Supplements Impacted:

Supplements 1, 2 and 3 are impacted.

1.3 Section 3.6.2.2.2 – SEND_QUEUING_MESSAGE message insertion

Problem Statement:

In Supplement 1, the pseudo code does not insert the message in the message queue (i.e. it does not transmit the message):

```
else
   -- There is sufficient space in the port's message queue to insert the
   -- message represented by MESSAGE_ADDR and LENGTH in the port's message queue;
   if (TIME_OUT is not infinite) then
      stop the time counter;
   end if;
   RETURN_CODE := NO_ERROR;
end if;
```

Resolution:

In Supplement 5, the pseudo code has been corrected to insert the message:

```
else
   -- There is sufficient space in the port's message queue to insert the
   -- message represented by MESSAGE_ADDR and LENGTH in the port's message queue;
   insert the message represented by MESSAGE_ADDR and LENGTH in the FIFO
    message queue of the specified port;
   if (TIME_OUT is not infinite) then
      stop the time counter;
   end if;
   RETURN_CODE := NO_ERROR;
end if;
```

Supplements Impacted:

Supplements 1, 2, 3 and 4 are impacted.

1.4 Section 3.8.2.4 – RAISE_APPLICATION_ERROR: module level recovery action

Problem Statement:

In Supplement 3, RAISE_APPLICATION_ERROR does not allow a module level recovery action, which is allowed by Figure 2.4:

```
if (this service is called by the error handler process) or (the error
   handler process is not created) then
   pass the message and error code to the Partition HM;
   take the recovery action described
   for the error code in the current Partition HM table;
else
```

Resolution:

In Supplement 5, RAISE_APPLICATION_ERROR is now consistent with Figure 2.4:

```
if (this service is called by the error handler process) or (the error
   handler process is not created) then
   pass the message and error code to the Partition HM or Module HM
    according to the partition's Multi-Partition HM table (see Section 2.4);
   take the Partition or Module recovery action described for the error code;
else
```

Supplements Impacted:

Supplements 3 and 4 are impacted.

1.5 Appendix D or D.1 – Ada 83 PROCESS_STATUS_TYPE 4 byte aligned

Problem Statement:

In Supplement 2, the Ada 83 PROCESS_STATUS_TYPE included a range that was not 4 byte aligned. Some compilers may have silently rounded the type to the 4 byte boundary, others may generate an error. NAME_TYPE, defined within ATTRIBUTES is only specified as a 30 character array. Changing this to a 32 character array resolves the alignment issue for all compilers:

```
for  PROCESS_STATUS_TYPE use record at mod 8;
   DEADLINE_TIME    at  0 range 0..63;
      CURRENT_PRIORITY at  8 range 0..31;
      PROCESS_STATE    at 12 range 0..31;
   ATTRIBUTES       at 16 range 0..62*8-1;
end record;
```

Resolution:

In Supplement 4, the Ada 83 PROCESS_STATUS_TYPE is now 4 byte aligned:

```
for  PROCESS_STATUS_TYPE use record at mod 8;
   DEADLINE_TIME    at  0 range 0..63;
   CURRENT_PRIORITY at  8 range 0..31;
   PROCESS_STATE    at 12 range 0..31;
   ATTRIBUTES       at 16 range 0..64*8-1;
end record;
```

Supplements Impacted:

Supplements 2 and 3 are impacted.

1.6 Appendix D or D.2 - ADA 95: Compiler enum type size

Problem Statement:

In Supplement 2, Ada compilers may or may not respect the intent of the “pragma Convention (C, EnumTypeName)” to ensure the size of enum types are 32 bits.

Resolution:

In Supplement 4, enumeration types in appendix D.2 now include “use” clauses that force the size to 32 bits. E.g. `Return_Code_Type'size use 32`

Supplements Impacted:

Supplements 2 and 3 are impacted.

1.7 Appendix D.2 - Processor_Core_Id_Type private/public

Problem Statement:

In Supplement 4, Processor_Core_Id_Type is specified as private. As a private type, this means the underlying implementation is hidden (could be an integer, could be a pointer to a data structure). For ARINC 653, this type is always an integer in the range 0 to X (where X is the number of cores allocated to a partition minus 1). As a private type, using the Initialize_Process_Core_Affinity API would result in an error (the Processor_Core_Id_Type is passed in). It should not be a private type.

```
    type Processor_Core_Id_Type is private;
    Core_Affinity_No_Preference : constant Processor_Core_Id_Type;
Private
```

ARINC SPECIFICATION 653, PART 1 – Page 152

APPENDIX D.2
ADA 95 INTERFACE SPECIFICATION

```
    Infinite_Time_Value : constant System_Time_Type := -1;
    type Processor_Core_Id_Type is new APEX_Integer;
    Core_Affinity_No_Preference : constant Processor_Core_Id_Type := -1;
end APEX;
```

Resolution:

In Supplement 5, Processor_Core_Id_Type is now public:

```
   type Processor_Core_Id_Type   is new APEX_Integer;

   Core_Affinity_No_Preference   : constant Processor_Core_Id_Type;

private
   Infinite_Time_Value           : constant System_Time_Type := -1;
   Core_Affinity_No_Preference   : constant Processor_Core_Id_Type := -1;
end APEX;
```

Supplement Impacted:

Only Supplement 4 is impacted.

1.8 Appendixes D, or D.1 and D.2 – ERROR_MESSAGE_SIZE_TYPE range

Problem Statement:

In Supplement 1, the size and alignment of ERROR_MESSAGE_SIZE_TYPE is inconsistent between Ada 83 and C:

```
type ERROR_MESSAGE_SIZE_TYPE is new APEX_TYPES.APEX_INTEGER
                    range 1..MAX_ERROR_MESSAGE_SIZE;
```

In Supplement 3, the size of ERROR_MESSAGE_SIZE_TYPE are inconsistent:

```
type ERROR_MESSAGE_SIZE_TYPE is new APEX_TYPES.APEX_INTEGER
      range 0..MAX_ERROR_MESSAGE_SIZE;
```
in Appendix D.1

```
subtype Error_Message_Size_Type is APEX_Integer range
      1..MAX_ERROR_MESSAGE_SIZE;
```
in Appendix D.2

Resolution:

In Supplement 4, the size of ERROR_MESSAGE_SIZE_TYPE are consistent in Appendix D.1 and Appendix D.2, but constrain the maximum size to one less than the intended maximum size. The following represents the intended specification for Appendix D.1:

```
type ERROR_MESSAGE_SIZE_TYPE  is new APEX_TYPES.APEX_INTEGER
   range 0..APEX_TYPES.APEX_INTEGER(MAX_ERROR_MESSAGE_SIZE);

type ERROR_MESSAGE_INDEX_TYPE is new APEX_TYPES.APEX_INTEGER
   range 0.. APEX_TYPES.APEX_INTEGER(MAX_ERROR_MESSAGE_SIZE-1);

type ERROR_MESSAGE_TYPE is array (ERROR_MESSAGE_INDEX_TYPE) of
   APEX_TYPES.APEX_BYTE;
```

and in Appendix D.2:

```
subtype Error_Message_Size_Type is APEX_Integer range
   0 .. Max_Error_Message_Size;

subtype Error_Message_Index_Type is APEX_Integer range
   0 .. (Max_Error_Message_Size-1);

type Error_Message_Type is
   array (Error_Message_Index_Type) of APEX_Byte;
```

Supplements Impacted:

Supplements 1 and 2, then 3, 4, and 5 are impacted.

1.9 Appendix H – XML Schema definitions

Problem Statement:

In Supplement 1, XML Schema definitions use different name spaces between Parts 1 and 2. This does not facilitate necessary dependencies between Parts 1 and 2.

Resolution:

Currently unresolved in the standard.

XML Schema files will be revised to:

a) Use a generic namespace (ARINC653): in between, a quick fix for this, and fix the XML error, is:

   To replace the 2 occurrences of “ARINC653/P1S5” by just “ARINC653” in each of the ARINC653Types.xsd and ARINC653TypesExtensible.xsd files.

b) Use standard (fixed) filenames

c) Embed an attribute in each file that defines the current version of that file.

Supplements Impacted:

Supplements 1, 2, 3, 4 and 5 are impacted.

2.0 ARINC 653 PART 2 ERRATA

This section contains descriptions of the known errata in ARINC 653 Part 2.

2.1 Section 3.12.2.3 – SET_DEFAULT_PROCESS_CORE_AFFINITY

Problem Statement:

In Supplement 3, a configuration check for affinity permission is missing from the pseudo code.

Resolution:

Currently unresolved in the standard.

The following error check should be included:

```
when (the configured set affinity permission for this
      partition is NOT_PERMITTED) =>
      RETURN_CODE := INVALID_CONFIG;
```

Supplements Impacted:

Supplements 3 and 4 are impacted.

2.2 Appendix E - MEMORY_BLOCK_STATUS_TYPE additional colon

Problem Statement:

In Supplement 1, MEMORY_BLOCK_STATUS_TYPE has colons inserted between the type and name. This violates C conventions and will not compile:

```
typedef
   struct {
      MEMORY_BLOCK_ADDR_TYPE : MEMORY_BLOCK_ADDR;
      MEMORY_BLOCK_MODE_TYPE : MEMORY_BLOCK_MODE;
      MEMORY_BLOCK_SIZE_TYPE : MEMORY_BLOCK_SIZE;
   } MEMORY_BLOCK_STATUS_TYPE;
```

Resolution:

In Supplement 4, colons have been removed:

```
typedef
   struct {
      MEMORY_BLOCK_ADDR_TYPE  MEMORY_BLOCK_ADDR;
      MEMORY_BLOCK_MODE_TYPE  MEMORY_BLOCK_MODE;
      MEMORY_BLOCK_SIZE_TYPE  MEMORY_BLOCK_SIZE;
   } MEMORY_BLOCK_STATUS_TYPE;
```

Supplements Impacted:

Supplements 1, 2, and 3 are impacted.

2.3 Appendix E - SAP_PORT_STATUS_TYPE additional colon

Problem Statement:

In Supplement 1, SAP_PORT_STATUS_TYPE has colons inserted between the type and name. This violates C conventions and will not compile:

```
typedef struct SAP_PORT_STATUS_TYPE
{
    MESSAGE_RANGE_TYPE  : NB_MESSAGE;
    MESSAGE_RANGE_TYPE  : MAX_NB_MESSAGE;
    MESSAGE_SIZE_TYPE   : MAX_MESSAGE_SIZE;
    PORT_DIRECTION_TYPE : PORT_DIRECTION;
    WAITING_RANGE_TYPE  : WAITING_PROCESSES;
    SAP_ADDRESS_TYPE    : CURRENT_DEST_ADDR;
    SAP_ADDRESS_TYPE    : CURRENT_SRC_ADDR;
} SAP_PORT_STATUS_TYPE;
```

Resolution:

In Supplement 4, colons have been removed:

```
typedef
   struct {
      MESSAGE_RANGE_TYPE   NB_MESSAGE;
      MESSAGE_RANGE_TYPE   MAX_NB_MESSAGE;
      MESSAGE_SIZE_TYPE    MAX_MESSAGE_SIZE;
      PORT_DIRECTION_TYPE  PORT_DIRECTION;
      WAITING_RANGE_TYPE   WAITING_PROCESSES;
      SAP_ADDRESS_TYPE     CURRENT_DEST_ADDR;
      SAP_ADDRESS_TYPE     CURRENT_SRC_ADDR;
   } SAP_PORT_STATUS_TYPE;
```

Supplements Impacted:

Supplements 1, 2, and 3 are impacted.

2.4 Appendix H – XML Schema definitions

Problem Statement:

In Supplement 1, XML Schema definitions use different name spaces between Parts 1 and 2. This does not facilitate necessary dependencies between Parts 1 and 2.

Resolution:

Currently unresolved in the standard.

XML Schema files will be revised to:

a) Use a generic namespace (ARINC653): in between, a quick fix for this, and fix the XML error, is:

   To replace the 2 occurrences of “ARINC653/P2S4” by just “ARINC653” in the ARINC653P2S4.xsd files.

b) Use standard (fixed) filenames

c) Embed an attribute in each file that defines the current version of that file.

Supplements Impacted:

Supplements 1, 2, 3, 4 and 5 are impacted.

3.0 ARINC 653 PART 4 ERRATA

This section contains descriptions of the known errata in ARINC 653 Part 4.

Note: Referring to note in Section 0.3 on Part 4 to Part 1 compatibility: Part 4 Base document is based on ARINC 653 Part 1 Supplement 3.

3.1 Appendix D.2 - ADA 95: Compiler enum type size

Problem Statement:

In Base document, Ada compilers may or may not respect the intent of the “pragma Convention (C, EnumTypeName)” to ensure the size of enum types are 32 bits.

Resolution:

Currently unresolved in the standard.

Enumeration types defined in appendix D.2 should include “use” clauses that force the size to 32 bits. E.g. `Return_Code_Type'size use 32`

Supplement Impacted:

Base document is impacted.

3.2 Appendixes D.1 and D.2 – ERROR_MESSAGE_SIZE_TYPE range

Problem Statement:

In Base document, the size and alignment of ERROR_MESSAGE_SIZE_TYPE is inconsistent between Ada and C, and incorrect in both Appendixes D.1 and D.2:

```
Max_Error_Message_Size : constant := ;

type ERROR_MESSAGE_SIZE_TYPE is new APEX_TYPES.APEX_INTEGER
          range ..MAX_ERROR_MESSAGE_SIZE;        in Appendix D.1

subtype Error_Message_Size_Type is APEX_Integer range
  1 .. Max_Error_Message_Size;                   in Appendix D.2
```

Resolution:

Currently unresolved in the standard.

The MAX_ERROR_MESSAGE_SIZE should be in both Appendixes D.1 and D.2:

```
Max_Error_Message_Size : constant := 128;
```

The following represents the intended specification for Appendix D.1:

```
type ERROR_MESSAGE_SIZE_TYPE  is new APEX_TYPES.APEX_INTEGER
   range 0..APEX_TYPES.APEX_INTEGER(MAX_ERROR_MESSAGE_SIZE);

type ERROR_MESSAGE_INDEX_TYPE is new APEX_TYPES.APEX_INTEGER
   range 0.. APEX_TYPES.APEX_INTEGER(MAX_ERROR_MESSAGE_SIZE-1);

type ERROR_MESSAGE_TYPE is array (ERROR_MESSAGE_INDEX_TYPE) of
   APEX_TYPES.APEX_BYTE;
```

and in Appendix D.2:

```
subtype Error_Message_Size_Type is APEX_Integer range
   0 .. Max_Error_Message_Size;

subtype Error_Message_Index_Type is APEX_Integer range
   0 .. (Max_Error_Message_Size-1);

type Error_Message_Type is
   array (Error_Message_Index_Type) of APEX_Byte;
```

Supplement Impacted:

Base document is impacted.

3.3 Appendix H – XML Schema definitions

Problem Statement:

In Base document, Part 4 refers to Part 1, and XML Schema definitions use different name spaces between Parts 1 and 2. This does not facilitate necessary dependencies between Parts 1 (and so 4) and 2.

Resolution:

Currently unresolved in the standard.

Part 4 referencing Part 1 in Appendix H, identical resolution to the Part 1 one should be considered for Part 4: XML Schema files should be revised to:

a) Use a generic namespace (ARINC653): in between, a quick fix for this, and fix the XML error, is:

   To replace the 2 occurrences of “ARINC653/P1S5” by just “ARINC653” in each of the ARINC653Types.xsd and ARINC653TypesExtensible.xsd files.

b) Use standard (fixed) filenames

c) Embed an attribute in each file that defines the current version of that file.

Supplement Impacted:

Base document is impacted.

ARINC Standard – Errata Report

(Insert the number, supplement level, date of publication, and title of the document with the error)

1. Document Title

2. Reference

Page Number: _____ Section Number: _____ Date of Submission: _____

3. Error

(Reproduce the material in error, as it appears in the standard.)

4. Recommended Correction

(Reproduce the correction as it would appear in the corrected version of the material.)

5. Reason for Correction (Optional)

(State why the correction is necessary.)

6. Submitter (Optional)

(Name, organization, contact information, e.g., phone, email address.)

Please return comments to **standards@sae-itc.org**

Note: Items 2-5 may be repeated for additional errata. All recommendations will be evaluated by the staff. Any substantive changes will require submission to the relevant subcommittee for incorporation into a subsequent Supplement.

[To be completed by IA Staff]

Errata Report Identifier:_____**Engineer Assigned:** _____

Review Status: _____

ARINC Standard Errata Form
June 2014