ARINC

AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE PART 1 – REQUIRED SERVICES

ARINC SPECIFICATION 653P1-3

PUBLISHED: November 15, 2010

AN ARINC DOCUMENT

Prepared by AEEC
Published by
AERONAUTICAL RADIO, INC.
2551 RIVA ROAD, ANNAPOLIS, MARYLAND 21401-7435

DISCLAIMER

THIS DOCUMENT IS BASED ON MATERIAL SUBMITTED BY VARIOUS PARTICIPANTS DURING THE DRAFTING PROCESS. NEITHER AEEC, AMC, FSEMC NOR ARINC HAS MADE ANY DETERMINATION WHETHER THESE MATERIALS COULD BE SUBJECT TO VALID CLAIMS OF PATENT, COPYRIGHT OR OTHER PROPRIETARY RIGHTS BY THIRD PARTIES, AND NO REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, IS MADE IN THIS REGARD.

ARINC INDUSTRY ACTIVITIES USES REASONABLE EFFORTS TO DEVELOP AND MAINTAIN THESE DOCUMENTS. HOWEVER, NO CERTIFICATION OR WARRANTY IS MADE AS TO THE TECHNICAL ACCURACY OR SUFFICIENCY OF THE DOCUMENTS, THE ADEQUACY, MERCHANTABILITY, FITNESS FOR INTENDED PURPOSE OR SAFETY OF ANY PRODUCTS, COMPONENTS, OR SYSTEMS DESIGNED, TESTED, RATED, INSTALLED OR OPERATED IN ACCORDANCE WITH ANY ASPECT OF THIS DOCUMENT OR THE ABSENCE OF RISK OR HAZARD ASSOCIATED WITH SUCH PRODUCTS, COMPONENTS, OR SYSTEMS. THE USER OF THIS DOCUMENT ACKNOWLEDGES THAT IT SHALL BE SOLELY RESPONSIBLE FOR ANY LOSS, CLAIM OR DAMAGE THAT IT MAY INCUR IN CONNECTION WITH ITS USE OF OR RELIANCE ON THIS DOCUMENT, AND SHALL HOLD ARINC, AEEC, AMC, FSEMC AND ANY PARTY THAT PARTICIPATED IN THE DRAFTING OF THE DOCUMENT HARMLESS AGAINST ANY CLAIM ARISING FROM ITS USE OF THE STANDARD.

THE USE IN THIS DOCUMENT OF ANY TERM, SUCH AS SHALL OR MUST, IS NOT INTENDED TO AFFECT THE STATUS OF THIS DOCUMENT AS A VOLUNTARY STANDARD OR IN ANY WAY TO MODIFY THE ABOVE DISCLAIMER. NOTHING HEREIN SHALL BE DEEMED TO REQUIRE ANY PROVIDER OF EQUIPMENT TO INCORPORATE ANY ELEMENT OF THIS STANDARD IN ITS PRODUCT. HOWEVER, VENDORS WHICH REPRESENT THAT THEIR PRODUCTS ARE COMPLIANT WITH THIS STANDARD SHALL BE DEEMED ALSO TO HAVE REPRESENTED THAT THEIR PRODUCTS CONTAIN OR CONFORM TO THE FEATURES THAT ARE DESCRIBED AS MUST OR SHALL IN THE STANDARD.

ANY USE OF OR RELIANCE ON THIS DOCUMENT SHALL CONSTITUTE AN ACCEPTANCE THEREOF “AS IS” AND BE SUBJECT TO THIS DISCLAIMER.

This document is published information as defined by 15 CFR Section 734.7 of the Export Administration Regulations (EAR). As publicly available technology under 15 CFR 74.3(b)(3), it is not subject to the EAR and does not have an ECCN. It may be exported without an export license.

2551 RIVA ROAD ANNAPOLIS, MARYLAND
21401-7435 USA

ARINC SPECIFICATION 653P1-3

AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE
PART 1 – REQUIRED SERVICES

Published: November 15, 2010

Prepared by the AEEC

Specification 653	Adopted by the AEEC Executive Committee	October 4, 1996

Summary of Document Supplements

Specification 653P1-1	September 16, 2003	October 16, 2003
Specification 653P1-2	October 4, 2005	March 7, 2006
Specification 653P1-3	October 6, 2010	November 15, 2010

A description of the changes introduced by each supplement is included at the end of this document.

FOREWORD

Aeronautical Radio, Inc., the AEEC, and ARINC Standards

ARINC organizes aviation industry committees and participates in related industry activities that benefit aviation at large by providing technical leadership and guidance. These activities directly support aviation industry goals: promote safety, efficiency, regularity, and cost-effectiveness in aircraft operations.

ARINC Industry Activities organizes and provides the secretariat for international aviation organizations (AEEC, AMC, FSEMC) which coordinate the work of aviation industry technical professionals and lead the development of technical standards for airborne electronic equipment, aircraft maintenance equipment and practices and flight simulator equipment and used in commercial, military, and business aviation. The AEEC, AMC, and FSEMC develop consensus-based, voluntary standards that are published by ARINC and are known as ARINC Standards. The use of ARINC Standards results in substantial benefits to the aviation industry by allowing avionics interchangeability and commonality and reducing avionics cost by promoting competition.

There are three classes of ARINC Standards:

a) ARINC Characteristics – Define the form, fit, function, and interfaces of avionics and other airline electronic equipment. ARINC Characteristics indicate to prospective manufacturers of airline electronic equipment the considered and coordinated opinion of the airline technical community concerning the requisites of new equipment including standardized physical and electrical characteristics to foster interchangeability and competition.

b) ARINC Specifications – Are principally used to define either the physical packaging or mounting of avionics equipment, data communication standards, or a high-level computer language.

c) ARINC Reports – Provide guidelines or general information found by the airlines to be good practices, often related to avionics maintenance and support.

The release of an ARINC Standard does not obligate any organization or ARINC to purchase equipment so described, nor does it establish or indicate recognition or the existence of an operational requirement for such equipment, nor does it constitute endorsement of any manufacturer’s product designed or built to meet the ARINC Standard.

In order to facilitate the continuous product improvement of this ARINC Standard, two items are included in the back of this volume:

An Errata Report solicits any corrections to the text or diagrams in this ARINC Standard.

An ARINC IA Project Initiation/Modification (APIM) form solicits any recommendations for addition of substantive material to this volume which would be the subject of a new Supplement.

ARINC SPECIFICATION 653
TABLE OF CONTENTS

APPENDICES

1.0 INTRODUCTION

1.1 Purpose

This document specifies the baseline operating environment for application software used within Integrated Modular Avionics (IMA) and traditional ARINC 700-series avionics.

The primary objective of this Specification is to define a general-purpose APEX (APplication/EXecutive) interface between the Operating System (O/S) of an avionics computer resource and the application software. Included within this specification are the interface requirements between the application software and the O/S and the list of services which allow the application software to control the scheduling, communication, and status information of its internal processing elements.

This Specification defines the data exchanged statically (via configuration) or dynamically (via services) as well as the behavior of services provided by the O/S and used by the application. It is not the intent of this specification to dictate implementation requirements on either the hardware or software of the system, nor is it intended to drive certain system-level requirements within the system which follows this standard.

The majority of this document describes the runtime environment for embedded avionics software. This list of services identifies the minimum functionality provided to the application software and is, therefore, the industry standard interface. It is intended for this interface to be as generic as possible, since an interface with too much complexity or too many system-specific features is normally not accepted over a variety of systems. The software specifications of the APEX interface are High-Order Language (HOL) independent, allowing systems using different compilers and languages to follow this interface.

This document is intended to complement ARINC Report 651. It is expected that this document will evolve to contain additional functionality and capability. Supplements to this document will be prepared as needed by the industry.

1.2 Scope

This document specifies both the interface and the behavior of the API services. Behavior is specified to the extent needed to describe functionality relevant to calling applications.

Where necessary, assumptions are made as to the support or behavior provided by the operating system and hardware. This should not be construed as a specification for the O/S or hardware. However, where the O/S or hardware does not coincide with the stated assumptions, the API behaviors specified herein may not match the actual behavior.

ARINC 653 is intended for use in a partitioned environment. In order to assure a high degree of portability, aspects of the partitioned environment are discussed and assumed. However, this specification does not define the complete system, hardware, and software requirements for partitioning nor does it provide guidance on proper implementation of partitioning, and in particular, robust partitioning. It must not be construed that compliance to ARINC 653 assures robust partitioning.

1.3 APEX Phases

This document defines phase 1 of the APEX interface specification. It is envisaged that the APEX interface will evolve over several phases from this kernel standard. The final APEX interface definition is expected to provide basic functionality for applications ranging from flight critical applications up to a full function O/S interface for demanding applications, such as database management and mass data storage and retrieval. It is a goal for later phases to be compatible with wider industry standards developed within the general purpose computing industry. It is a goal to maintain evolutionary compatibility through subsequent phases, such that earlier phases may be subsets of later phases. Figure 1.1 depicts the phased approach.

Phase 1 is targeted for critical systems and contains a sufficient subset of services needed for applications to accomplish their function and to support certification. The interface to the Ada run-time system will not be defined, and Ada tasking will not be supported. The objective is to provide an interface to the application software only.

The APEX interface definition for non-critical systems is deferred until later phases. However, common operational requirements for these systems have been considered in the current phase in order to establish a consistent evolutionary framework for both critical and non-critical systems.

ARINC 653-1 supercedes the original release of the standard (1997). It provides refinement and clarification to the phase 1 standard.

ARINC 653 Part 1 supersedes ARINC 653-1. It provides minor clarifications and has been reformatted as a result of the division of ARINC 653 into three parts.

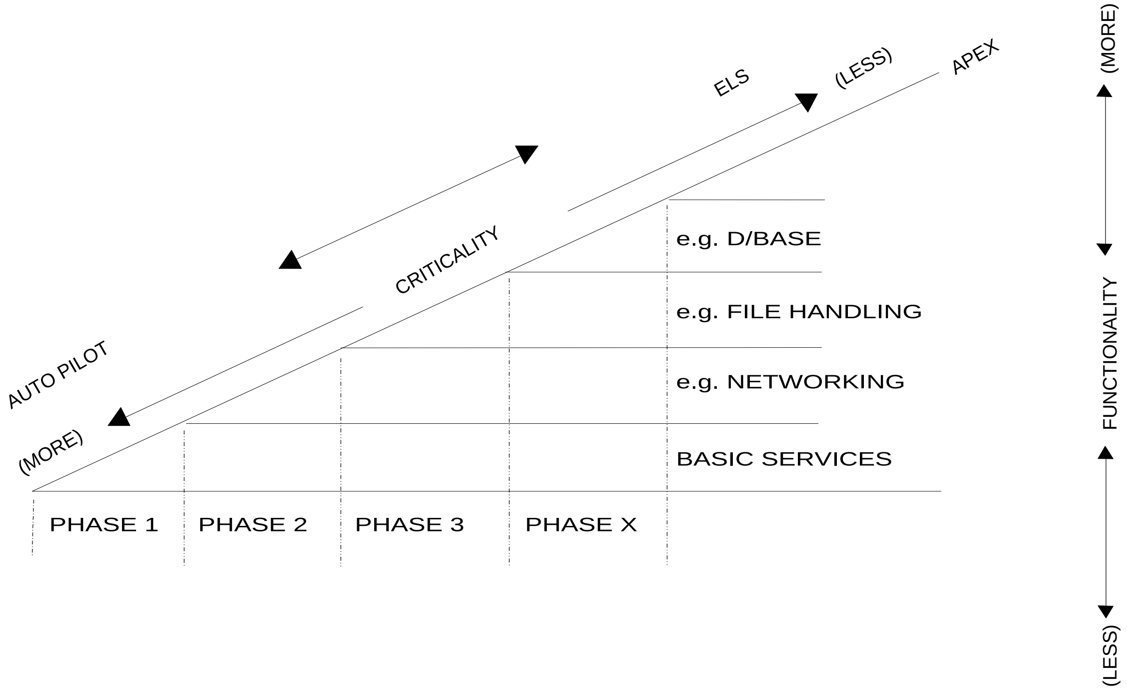


Figure 1.1 - Example of APEX Interface Phased Approach

1.4 ARINC Specification 653 Basic Philosophy

1.4.1 IMA Partitioning

One purpose of a core module in an IMA system is to support one or more avionics applications and allow independent execution of those applications. This can be correctly achieved if the system provides partitioning, i.e., a functional separation of the avionics applications, usually for fault containment (to prevent any partitioned function from causing a failure in another partitioned function) and ease of verification, validation and certification.

The unit of partitioning is called a partition. A partition is basically the same as a program in a single application environment: it comprises data, its own context, configuration attributes, etc. For large applications, the concept of multiple partitions relating to a single application is recognized.

1.4.2 Software Decomposition

The software that resides on the hardware platform consists of:

- Application partitions are the portions of software specific to avionics applications supported by the core module. This software is specified, developed and verified to the level of criticality appropriate to the avionics application. ARINC 653 Application partitions are subject to robust space and time partitioning and are restricted to using only ARINC 653 calls to interface to the system.
- An O/S kernel that provides the API and behaviors defined within this specification and supports a standard and common environment in which application software executes. This may include hardware interfaces such as device drivers and built-in-test functions.
- System partitions are partitions that require interfaces outside the scope of APEX services, yet, constrained by robust spatial and temporal partitioning. These partitions may perform functions such as managing communication from hardware devices or fault management schemes. System partitions are optional and are specific to the core module implementation.
- System specific functions may include hardware interfaces, such as device drivers, down loading, debug and built-in-test functions.

The APEX interface, between the application software and the O/S, defines a set of facilities which the system will provide for application software to control the scheduling, communication, and status information of its internal processing elements. The APEX interface can be seen from the viewpoint of the application software as a high order language (HOL) specification and from the O/S viewpoint as a definition of parameters and entry mechanisms (e.g., trap calls). APEX may include a layer that translates from the HOL specification into the appropriate entry mechanism. This translation is directly dependent on the O/S implementation, hardware platform, and probably also on the compiler used for application software. If library routines are used, they may need to be included within the application code to preserve robust partitioning.

This specification provides the definition of APEX interface services. The additional interface available to system partitions is not detailed here. There are two reasons for not including the System Partition interface:

1. This standard covers avionics application to platform interfaces only. Therefore, the definition of the System Partition interface is out of scope.
2. The System Partition interface is likely to be specific to the O/S or platform hardware (for example device drivers) and, therefore, cannot be generically defined in this standard.

1.4.3 Goals

The APEX interface provides a common logical environment for the application software. This environment enables independently-produced applications to execute together on the same hardware.

The principal goal of the APEX interface is to provide a general-purpose interface between the application software and the O/S within IMA. The concept of an O/S that is common to different hardware implementations is not new. However, the use of this technique for embedded systems is new; because in the past system designers have considered customized solutions to be more reliable and efficient for real-time applications.

The current level of sophistication of software/hardware technologies is such that the advantages of the use of a generalized system will exceed the potential disadvantages. By the specification and adoption of an industry-wide APEX interface, the ability to develop a flexible general-purpose computer is available to the avionics industry.

Standardization of the interface will allow for the use of application code, hardware, and O/S from a wide range of suppliers, encouraging competition and allowing reduction in the costs of system development and ownership. To ensure that the APEX interface brings maximum benefit to the industry, the following goals are established:

a. The APEX interface provides the minimum number of services consistent with fulfilling IMA requirements. The interface will consequently be easy for application software developers to efficiently utilize, assisting them in production of reliable products.
b. The APEX interface is extendable to accommodate future system enhancements. The O/S will realistically be subject to expansion in the future, and so the APEX interface should be easily extended while retaining compatibility with previous versions of the software.
c. The APEX interface should satisfy the common real-time requirements of Ada 83, Ada 95, and CIFO. These different models should use the underlying services provided by the APEX interface in accordance with their certification and validation criticality level.
d. The APEX interface decouples application software from the actual processor architecture. Therefore, hardware changes will be transparent to the application software. The APEX interface allows application software to access the executive’s services but insulates the application software from architectural dependency.
e. The APEX interface specification is language independent. This is a necessary condition for the support of application software written in different high-level languages or application software written in the same language but using different compilers. The fulfillment of this requirement allows flexibility in the selection of the following resources: compilers, development tools, and development platforms.

1.4.4 Expected Benefits

The following objectives are expected to be achieved with the APEX interface:

a. Portability: The APEX interface facilitates portability of the software. It is desirable for the application software developed for a particular aircraft to be transported to other aircraft types with minimal recertification effort. By removing language and hardware dependence, the APEX interface achieves this objective.
b. Reusability: The APEX interface allows the production of reusable application code for IMA systems. The APEX interface will reduce the amount of customizing required when a component is reused.
c. Modularity: The APEX interface provides the benefits of modularity when developing application software. By removing hardware and software dependencies, the APEX

interface reduces the impact on application software from modifications to the overall system.

d. Integration of Software of Multiple Criticalities: The APEX interface supports the ability to co-locate application software of different levels of criticality.

1.5 Implementation Guidelines

The primary goals of the APEX interface are first to define a set of common procedure calls which the writer of the application software may use, and secondly, to define a set of facilities which the O/S will provide.

At the highest level, the two goals do not conflict. However, as the APEX interface is merely a definition of facilities, the provider of the O/S may choose any mechanism to provide these facilities. The most direct way is to define a set of procedure calls.

The provider of the O/S is expected to produce a software package which consists of a set of procedures (functions, subroutines, etc.) which access the low level facilities of the machine. These procedures are then called via the APEX interface.

Implementation of the APEX interface does not require a definition of the way in which the compiler handles the calls. Further, there is no necessity for the application software and O/S to use the same language. The actual mechanism, at the machine level, by which the functions are called is a property of the compiler used by the O/S software.

Numerous alternatives exist for APEX interface implementation. Three alternatives are mentioned here. One alternative is a special or prescribed compiler for the IMA software. Another is an assembler within the application code so that the O/S functions may be accessed in the required way. A third is a set of procedures which perform the translation from the application code calls to O/S calls.

The provision of a special compiler is the most expensive option and will lead to inflexibility and potential problems with future support.

To allow an assembler within the application software is not satisfactory for the reason that there will be great difficulty in achieving portability.

With the last option, the application software may be made implementation independent if a translation interface to the O/S is provided. This can be seen as being separate from the application software. In this case, the application code itself is portable and reusable.

The concept of an “active” interface has been applied in the definition (by the International Standards Organization) of the Open System Interconnection communications. Most relevant to the APEX interface is the communications layer known as the “Transport” layer. This allows software dealing with hardware functions to provide common facilities for application software.

It may be necessary to provide a translation layer if the procedure-calling mechanism used by the application code differs from that required by the O/S.

The application software writer should provide software which is hardware independent and makes no assumptions about the implementation of the O/S. Although the APEX interface definition is expected to permit hardware independence, certain aspects of the O/S implementation may make this difficult to achieve.

The provider of the O/S should be knowledgeable about the processor architecture and the services required of the APEX interface. The implementation of the O/S may vary from one processor architecture to another. Further, the implementation of the APEX services may be provided in any way the O/S writer chooses.

1.5.1 Portability

Application software portability can be achieved if the software is written without regard to hardware implementations and configurations. Therefore, the application software should use the standard services whenever available. The application software should also avoid other implementation dependencies, for example, the application software of a partition should not be dependent on the location (i.e., same module or same cabinet) of other partitions.

1.5.2 Language Considerations

The Ada language is recommended and supported by airlines for application software development, but other languages (such as C) could also be used. Parts of the application software of a partition which directly use APEX services, and a HOL specification of the APEX interface, should be written in the same language. Therefore, it is necessary to define as many distinct HOL specifications for the APEX interface as there are required languages for application software development. These specifications should meet certain requirements.

Each specification should form a complete definition of the APEX interface from the application's viewpoint.

All specifications should provide services with the same semantics independently of the language. This requirement allows support of multiple specifications by the same O/S.

There may be differences in the syntax of services specified in distinct languages due to differences in typing and declaration scopes of the languages.

COMMENTARY

> Some significant and unavoidable differences are visible in the syntax of services which are specified in both a language with strong typing like Ada, and another language with less strong typing facilities like C. With the former, subprogram units are encapsulated into packages, while the latter does not provide an equivalent facility. Subprogram parameter passing techniques are also quite different. Ada requires that a mode be specified for each parameter. The compiler then chooses to pass parameters either by value or reference; it also adds implicit parameters sometimes. The full list of parameters together with the user-required passing method (value or reference) should be specified in C.

Each specification should be compiler independent.

All the applications executing on a particular **core module** may not necessarily use the same specification (i.e., in the same language).

1.6 Document Overview

ARINC Specification 653 is presently organized into three parts, as follows:

Part 1 Avionics Software Standard Interface (this document)

Part 2 APEX Extensions

Part 3 APEX Compliance Test Procedure

Part 1 constitutes the basic requirements and guidance for an ARINC 653 compliant O/S API. Part 1 is organized into five sections and a number of appendices, as follows:

- Section 1 Provides overview material. The Introduction section describes the purpose of this document and provides an overview of the basic philosophy behind the development of the APEX interface
- Section 2 Provides conceptual and background information about the services specified in the document
- Section 3 Provides the API specification and pseudo code for each of the services
- Section 4 Discusses compliance to Part 1 of ARINC 653
- Section 5 XML Configuration Specification
- Appendix A Glossary
- Appendix B Acronyms
- Appendix C **(deleted by Supplement 3)**
- Appendix D Ada83 and Ada95 Language Interface Specification
- Appendix E The C Language Interface Specification
- Appendix F (deleted by Supplement 1)
- Appendix G Graphical view of XML Configuration Schema Types
- Appendix H **XML Schema Types**
- **Appendix I Example XML Schema and its Corresponding XML Data Instance**

The Introduction section describes the purpose of this document and provides an overview of the basic philosophy behind the development of the APEX interface.

The System Overview section defines the assumptions pertinent to the hardware and software implementation within the target system, while it also identifies the assumed requirements specific to the O/S and application software. It identifies the basic concepts of the execution environment of the system, providing an overall understanding of the interactions between the individual elements within the system.

The Service Requirements section identifies those fundamental features which the application software needs in order to control the operation and execution of its processes. These requirements neither favor one particular type of application nor advocate any specific designs of the application software. This section also identifies the actual list of service requests which satisfy the individual requirements. These requests form the basis of this interface standard.

The Compliance section discusses necessary consideration for an O/S or application to be compliant to the APEX interface.

The XML Configuration section discusses the use of XML for defining the contents of the configuration.

Part 2 defines optional extended services. Further discussion concerning the organization of Part 2 can be found in Part 2.

Part 3 contains a compliance test procedure used to establish compliance to ARINC 653 Part 1. Further discussion concerning the organization of Part 3 can be found in Part 3.

1.7 Relationship to Other Standards

This document considers the functionality of the interfaces developed by the working groups described below. No single interface definition satisfies all of the requirements of an IMA architecture. Therefore, the APEX interface definition makes collective use of the documents produced by these groups.

1.7.1 ARTEWG/CIFO

The ARTEWG (Ada RunTime Environment Working Group) is sponsored by the ACM SIGAda (Association for Computing Machinery, Special Interest Group on Ada). The goals of the ARTEWG are to establish conventions, criteria, and guidelines for Ada runtime environments that facilitate the reusability and transportability of Ada program components, improve the performance of those components, and provide a framework which can be used to evaluate Ada runtime systems. ARTEWG has produced several documents, including the Catalogue of Interface Features and Options for the Ada Runtime Environments (CIFO).

The CIFO is a catalogue of proposed interfaces to runtime environments. This document proposes and describes a common set of user-runtime environment interfaces from a user’s perspective. These interfaces are described as entries. Common interfaces are clearly needed to make the development of high-quality software practical for the full range of applications that Ada was intended to serve, especially the domain of embedded real-time systems.

1.7.2 ISO/ExTRA

ISO-IEC/JTC1/SC22/WG9 is the working group which addresses Ada standardization issues. The main goal of the WG9 Real Time Rapporteur Group was to provide an annotated specification of Ada (ISO/IEC 8652) packages that provide the services necessary for hard real-time systems, typically complex life-critical systems with hard deadlines. The result is the ISO/IEC TR 11735 document also named ExTRA (Extensions for Real Time Ada).

The purpose of the ISO/IEC 11735 is to define a standard Ada library for hard real-time systems to support portability at the source level. This is intended for application software developers as well as for Ada real-time executive developers. The library unit interfaces described in this document are based on CIFO 3.0 and the ExTRA project which defined and provided uniform services for hard real-time applications in avionics, aerospace, and embedded Ada software systems.

1.7.3 Ada 95

The goal of the ISO-IEC/JTC1/SC22/WG9 Ada 9X Project was to revise Ada ISO/IEC 8652:1987, which is also ANSI/MIL-STD-1815A (Ada 83), to reflect the current essential requirements with minimal negative impact and maximum positive impact to the Ada community. Another goal was to coordinate this revision with the international community to ensure publication of the Ada 9X document as an ISO standard. The result is the ANSI/ISO/IEC 8652:1995 document also named Ada 95.

1.7.4 POSIX

The Portable Operating System Interface for computer environments (POSIX) is a Unix-compatible system environment. POSIX is intended to promote portability of Unix applications across the various Unix-derived environments. It is a standard of the Institute of Electrical and Electronics Engineers (IEEE). Although originated to refer to IEEE Std 1003.1-1988, the name POSIX more correctly refers to a family of related standards: IEEE 1003.n and the parts of international standard ISO-IEC 9945. ISO-IEC/SC22/WG15 is the working group which addresses the POSIX standardization issues. Two POSIX areas of standardization of interest to the APEX interface are the real-time extensions (1003.1b, 1003.1c, etc.) and the language bindings (1003.5, 1003.20, etc.).

1.7.5 Extensible Markup Language (XML) 1.0

The Extensible Markup Language (XML) is a subset of SGML (ISO 8879) that is completely described in http://www.w3.org/TR/2000/REC-xml-20001006. The XML standard is maintained by the World Wide Web Consortium (WC3). XML is used herein to specify the scheme for defining ARINC 653 configuration data.

1.8 Related Documents

The latest version of the following documents apply.

1.8.1 ARINC Report 651

ARINC Report 651: *Design Guidance For Integrated Modular Avionics* is a top-level design guide for the design and implementation of modular avionics systems. ARINC Specification 653 is a result of specific requirements identified within ARINC Report 651.

1.8.2 ARINC Report 652

ARINC Report 652: *Guidance For Avionics Software Management* is a top-level design guide for managing the development and maintenance of avionics software. It provides guidance for the design and implementation of avionics software for traditional ARINC 700-Series equipment and Integrated Modular Avionics (IMA).

1.8.3 ARINC Report 613

ARINC Report 613: *Guidance For Using The Ada Programming Language In Avionic Systems* provides guidance on the use of the Ada programming language in commercial avionics applications. It addresses the use of the Ada programming language in the development, testing, and maintenance of digital avionics for the commercial aviation industry.

1.8.4 ARINC Specification 629

(The material was deleted by Supplement 1 and section renumbered by Supplement 2).

1.8.5 ARINC Specification 659

(The material was deleted by Supplement 1 and section renumbered by Supplement 2).

1.8.6 RTCA DO-178/EUROCAE ED-12

RTCA DO-178 / EUROCAE ED-12: *Software Considerations in Airborne Systems and Equipment Certification* contains guidance and standards concerning software development and certification issues for the operational safety of the aircraft.

RTCA DO-248B/ED-94B: *Final Report for Clarification of DO-178B, Software Considerations in Airborne Systems and Equipment Certification* has been created to provide clarification of the guidance material in DO-178B/ED-12B. Included are errata, Frequently Asked Questions (FAQ), and Discussion Papers. Of particular relevance to ARINC 653 is Discussion Paper DP #14, Partitioning Aspects of DO-178B/ED-12B, where the concept of robust partitioning is introduced and defined.

1.8.7 RTCA DO-255 / EUROCAE ED-96

This document contains the Requirements Specification for an Avionics Computer Resource (ACR). The ACR provides shared computation resources, core software and signal conditioning necessary to interface with a variety of aircraft systems.

2.0 SYSTEM OVERVIEW

2.1 System Architecture

This interface specification has been developed for use with an Avionics Computer Resource (ACR). Its implementation may be Integrated Modular Avionics (IMA) or **federated avionics within a Line Replaceable Unit (LRU). The system architecture supports partitioning in accordance with the IMA philosophy.**

Figure 2.1 illustrates a module architecture. Note that this figure is not intended to imply layering that must be represented in an O/S implementation.

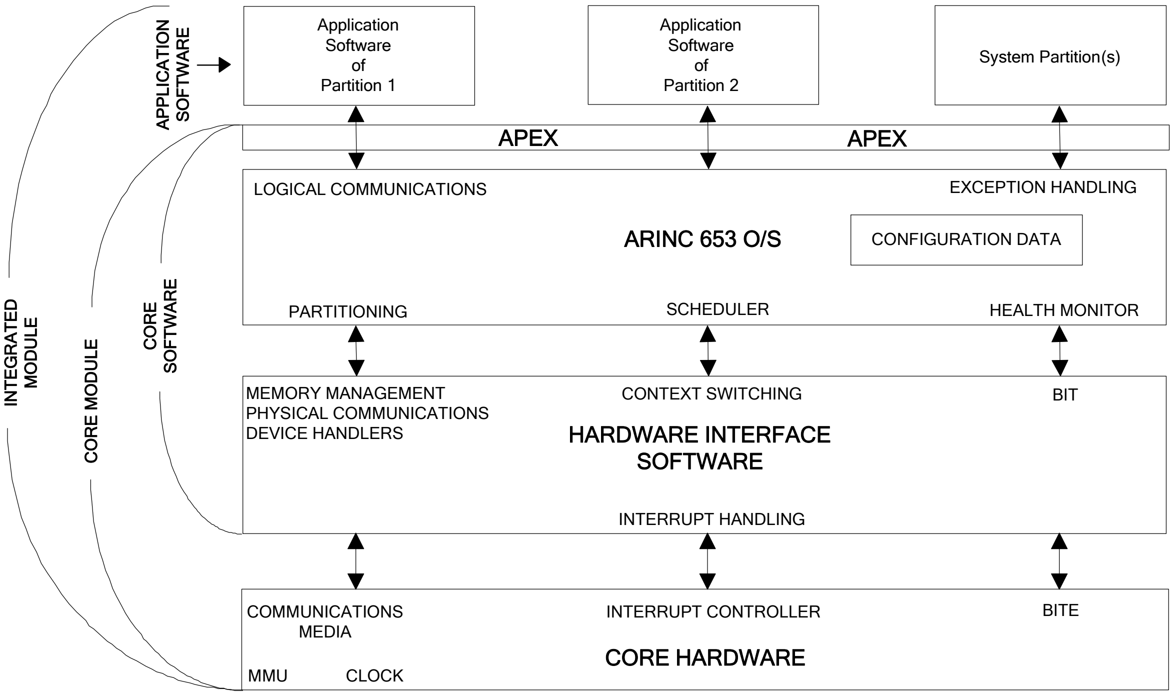


Figure 2.1 – Example Module Architecture

COMMENTARY

> **The term "Core Software" used in Figure 2.1 is generally referred to as "O/S" in this document, whereas the term "Integrated Module" is typically abbreviated to "Module".**

Each core module can contain one or more individual processors. The core module architecture has influence on the O/S implementation, but not on the APEX interface used by the application software of each partition. Application software should be portable between core modules and between individual processors of a core module without modifying its interface with the O/S.

COMMENTARY

> **In the context of this document, when there are multiple processors on a common hardware element, each processor or a set of processors that hosts a single ARINC 653 API execution context (i.e., execute a single partition schedule at any given time) is considered to constitute a separate core module.**

To achieve the required functionality and conform to specified timing constraints, the constituent processes of a partition may operate concurrently. The O/S provides services to control and support the operational environment for all processes within a partition. In particular, concurrency of operation is provided by the partition-level scheduling model.

The underlying architecture of a partition is similar to that of a multitasking application within a general-purpose mainframe computer. Each partition consists of one or more concurrently executing processes, sharing access to processor resources based upon the requirements of the application. All processes are uniquely identifiable, having attributes that affect scheduling, synchronization, and overall execution.

To ensure portability, communication between partitions is independent of the location of both the source and destination partition. An application sending a message to, or receiving a message from, another application will not contain explicit information regarding the location of its own host partition or that of its communications partner. The information required to enable a message to be correctly routed from source to destination is contained in configuration tables that are developed and maintained by the system integrator, not the individual application developer. The System Integrator configures the environment to ensure the correct routing of messages **between partitions within a module and between modules**. How this configuration is implemented is outside the scope of this standard.

2.2 Hardware

In order to isolate multiple partitions in a shared resource environment, the hardware should provide the O/S with the ability to restrict memory spaces, processing time, and access to I/O for each individual partition.

Partition timing interrupt generation should be deterministic. Any interrupts required by the hardware should be serviced by the O/S. Time partitioning should not be disturbed by the use of interrupts.

COMMENTARY

> As an example, ARINC 659, Backplane Data Bus, specifies a table driven protocol consisting of a cyclic series of message windows of predefined lengths. Use of this backplane bus directly impacts the partition scheduling model. Avionics equipment may include other types of backplane bus or none at all (e.g., they may communicate with their environment via ARINC 429). Systems that support multiple partitions should internally generate interrupts at regular time intervals to schedule those partitions.

There are several basic assumptions made about the processor:

1. The processor provides sufficient processing to meet worst-case timing requirements.
2. The processor has access to required I/O and memory resources.
3. The processor has access to time resources to implement the time services.
4. The processor provides a mechanism to transfer control to the O/S if the partition attempts to perform an invalid operation.
5. The processor provides atomic operations for implementing processing control constructs. These atomic operations will induce some jitter on time slicing. Also, atomic operations are expected to have minimal effect on scheduling.

2.3 System Functionality

This section describes the functionality to be provided by the O/S and its interface with the application software.

At the core module level, the O/S manages partitions (partition management) and their interpartition communication, the latter being conducted either within or across module boundaries. At the partition level, the O/S manages processes within a partition (process management) and communication between the constituent processes (intra-partition communication). The O/S may therefore be regarded as multi-level according to its scope of operation. O/S facilities (e.g., scheduling, message handling) are provided at each level, but differ in function and content according to their scope of operation. Definition of these facilities therefore distinguishes between the level at which they operate.

At any time instance, the module is in initialization, operational, or idle state. On power-up, the module begins execution in the initialization state. On successful completion of initialization, the module enters the operational state. During operational state, the O/S manages the partitions, processes and communications. The module continues in the operational state until power is removed, or the module health monitoring function commands the module to reinitialize or to enter idle state.

COMMENTARY

> **The O/S implementer may further refine module states in support of implementing module health monitoring (see Section 2.4).**

2.3.1 Partition Management

Central to the ARINC 653 philosophy is the concept of partitioning, whereby the **applications resident in a module** are partitioned with respect to space (memory partitioning) and time (temporal partitioning). A partition is therefore a program unit of the application designed to satisfy these partitioning constraints.

The O/S supports robust partitioning. A robustly partitioned system allows partitions with different criticality levels to execute in the same module, without affecting one another spatially or temporally.

The portion of the O/S that operates at the core module level is responsible for enforcing partitioning and managing the individual partitions within the module. This portion of the O/S should be isolated and protected against failures of any software (e.g., application, O/S, language support libraries, etc.) executing in the context of a partition.

Partitions are scheduled on a fixed, cyclic basis. To assist this cyclic activation, the O/S maintains a major time frame of fixed duration, which is periodically repeated throughout the module's runtime operation. Partitions are activated by allocating one or more partition windows within this major time frame. Each partition window is defined by its offset from the start of the major time frame and expected duration. The order of partition activation is defined **by the system integrator** using configuration tables. This provides a deterministic scheduling methodology whereby the partitions are furnished with a predetermined amount of time to access processor resources.

A module may contain several partitions running with different periods. The major time frame is defined by the multiple of the least common multiple of all partition periods in the module. Each major time frame contains identical partition windows. The periodic requirement of each partition **in**

a module must be satisfied by the appropriate size and frequency of partition windows within the major time frame.

Within a partition window, the partition's processes are managed as described in Section 2.3.2.3.

COMMENTARY

> Temporal partitioning is influenced by the O/S overhead. **Inter-module communications** acknowledgements and time-outs may interrupt one partition even though the events relate to a different partition. As a result, **the time duration allocated for use by an application may be impacted.**

Each partition has predetermined areas of memory allocated to it. These unique memory areas are identified based upon the requirements of the individual partitions, and vary in size and access rights.

Configuration of all partitions throughout the whole system is expected to be under the control of the system integrator and maintained with configuration tables. The configuration table for the partition schedule will define the major time frame and describe the order of activation of the partition windows within that major time frame.

Partition management services are available to the application software **for setting a partition's operating mode and to obtain a partition's status**.

2.3.1.1 Partition Attribute Definition

In order for partitions to be supported by a core module, a set of unique attributes will be defined for each partition. These attributes enable the **O/S to control and maintain each** partition's operation.

Partition attributes are as follows:

FIXED ATTRIBUTES **(defined as part of the ARINC 653 XML-schema types)**

1. Identifier - **defines a numerical identity for the partition. It is uniquely defined at least on a core module basis. It may be used by the O/S to facilitate partition activation, message routing, and debug support. It may also be used by the application, for example, as a field within a maintenance message.**
2. **Name - defines a string identity for the partition.**
3. Memory Requirements - define memory bounds of the partition, with appropriate code/data segregation. **The memory requirements may include separate specification of the process stack sizes. It is O/S implementation dependent whether definition of process stack sizes are required as separate memory requirements or are included in a data memory allocation.**
4. **Partition** Period - **the required time interval at which the partition must be activated in order to satisfy its operational requirements.**

COMMENTARY

> **Typically, the partition period is the greatest common factor of the process periods within a partition. If the process periods are harmonic, then this is the period of the highest-rate process in a partition. If there are no periodic processes in a partition, then the period is based on the minimum execution frequency required by an application to satisfy its performance requirements.**

The partition period is used as an input in determining offsets and durations of partition windows.

5. **Partition Duration - the amount of execution time required by the partition within one partition period.**

COMMENTARY

This is used as an input in determining offsets and durations of partition windows.

6. **Inter-Partition** Communication Requirements **(Ports)** - denote those partitions and/or devices with which the partition communicates.
7. Partition Health Monitor Table - denotes **HM actions to be performed on detection of failures assigned to the partition**.

FIXED ATTRIBUTES (not included in the ARINC 653 XML-schema types)

1. Entry Point (i.e., partition initialization) - denotes partition **start/**restart address.
2. System Partition – denotes the partition is a system partition **(i.e., is not restricted to using ARINC 653 services to interact with the O/S)**.

VARIABLE ATTRIBUTES **(defined and controlled during run-time)**

1. Lock Level – denotes the current lock level of the partition **(see Section 2.3.2.6)**.
2. Operating Mode – denotes the partition's **execution state (see Section 2.3.1.4)**.
3. Start Condition – denotes the reason the partition is started.

2.3.1.2 Partition Control

The O/S starts the application partitions when **the O/S enters operational state**. The resources used by each partition (channels, processes, queues, semaphores, events, etc.) are specified at system build time. The corresponding objects **(i.e., data structures) are created during partition's** initialization phase, and then the partition enters NORMAL mode. The **application** is responsible for invoking the appropriate APEX calls to transition **the partition** from one operational mode to another. The operational mode of one partition is independent of the operational mode of other partitions. Thus, some partitions may be in the COLD_START mode while other partitions are in the NORMAL or WARM_START mode. The HM function can restart, or set to IDLE mode, a single partition, multiple partitions, or the entire module in response to a fault.

2.3.1.3 Partition Scheduling

Scheduling of partitions is strictly deterministic over time. Based upon the configuration of partitions **within a module**, overall resource requirements/availability, and specific partition requirements, a time-based activation schedule is generated that identifies the partition windows allocated to the individual partitions. Each partition is then scheduled according to its respective partition windows.

The schedule is fixed for the particular configuration of partitions **within a module**. The main characteristics of the partition scheduling model are:

1. **From the application developer perspective the** scheduling unit is a partition.
2. Partitions have no priority.
3. The **partition** scheduling algorithm is predetermined, repetitive with a fixed periodicity **(major time frame)**, and is configurable **via the configuration tables** only. At least one

partition window **should be** allocated to each partition during each cycle **of the major time frame**.

The following are used in specifying the partition schedule within the module configuration:

a. **Partition Time Window: An uninterrupted interval of execution time provided to a partition within a partition schedule. The interval is defined by Partition Time Window Duration and Partition Time Window Offset.**

b. **Partition Time Window Duration: The quantity of execution time in the partition time window.**

c. **Partition Time Window Offset: Time between the start of the major time frame and the activation of the partition time window.**

COMMENTARY

The Partition Time Window is used in defining the schedule and is the result of integration of all partition period and duration requirements.

d. **Periodic Processing Start: A point in a partition schedule aligned to the beginning of a partition's window where a given partition's periodic process scheduling is permitted to start.**

COMMENTARY

There may be more than one Periodic Processing Start point for a partition defined in a partition schedule. The first Periodic Processing Start point encountered following the partition switching to NORMAL MODE will be used to start periodic process scheduling.

4. **The O/S exclusively controls the allocation of the processor resources to the partition.**

COMMENTARY

The use of an inter-module communication mechanisms such as the ARINC 659 backplane data bus may require the system integrator to schedule the message exchanges among all partitions hosted by core modules sharing the mechanism. Synchronizing partition message generation/processing within the partition's activation period implies partition scheduling driven by the communication mechanism. Altering an established inter-module communication schedule could potentially affect partition scheduling on all of the core modules sharing the mechanism.

2.3.1.4 Partition Operating Modes

The current execution state of the partition (idle, cold start, warm start, or normal) represents the partition's operating mode. The SET_PARTITION_MODE service allows the **application associated with a** partition to request a change to its operating mode. The Health Monitor, through the health monitoring configuration **tables**, can also **request changes to each partition's operating** mode. The current **operating** mode of the partition **can be determined by using** the GET_PARTITION_STATUS service.

Partition **operating** modes and their state transitions are shown in **Figure 2.3.1.4.**

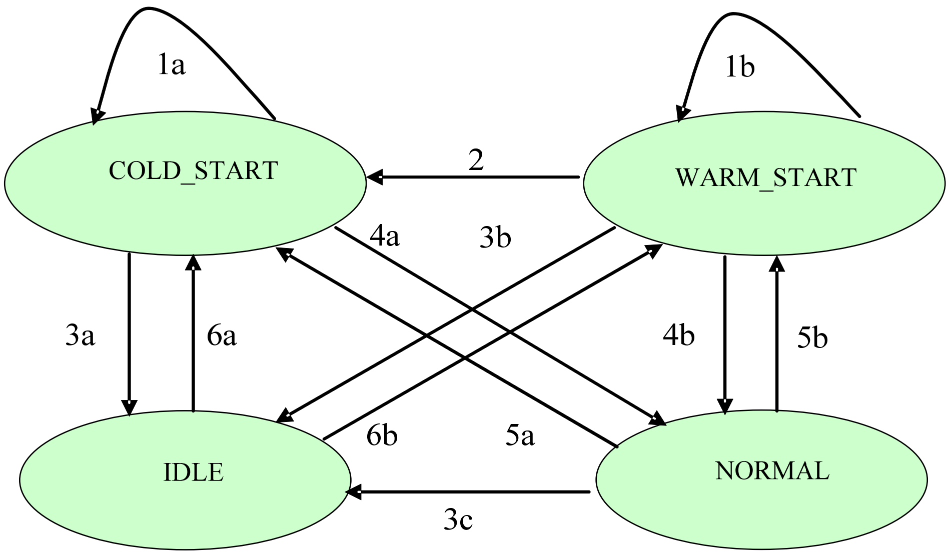


Figure 2.3.1.4 – Partition Operating Modes and Transitions

These modes are described in Section 2.3.1.4.1, and the illustrated transitions are described in Section 2.3.1.4.2. Process scheduling varies between the modes. Every partition is allocated a main process (not an ARINC process, i.e., there is no process identifier) that is utilized during COLD_START and WARM_START. The main process is the default process of the application, and is the only process eligible for scheduling during the COLD_START and WARM_START partition operating modes.

2.3.1.4.1 Partition Operating Modes Description

IDLE: In this mode, **no application defined partitions are executing within the partition's** allocated partition windows. The partition is not initialized (e.g., none of the ports associated to the partition are initialized), no processes are executing, but the **partition** windows allocated to the partition are **preserved**.

NORMAL: In this mode, **the partition's initialization phase is complete and** the **partition's** process scheduler is active. All processes that have been created and those that are in the ready state are **eligible** to run.

COLD_START: In this mode, the **partition's** initialization phase is in progress. Preemption is disabled with **LOCK_LEVEL > 0** (process scheduling is inhibited) and the **associated application** is executing its respective initialization code.

WARM_START: In this mode, the **partition's** initialization phase is in progress. Preemption is disabled with **LOCK_LEVEL > 0** (process scheduling is inhibited) and the **associated application** is executing its respective initialization code. This mode is similar to the COLD_START, but the initial environment (the hardware context in which the partition starts) may be different (e.g., no need for copying code from Non Volatile Memory to RAM).

2.3.1.4.1.1 COLD_START and WARM_START Considerations

The differences between COLD_START and WARM_START depend mainly on hardware capabilities and system specifications. For example, a power interrupt does not imply to start automatically in the COLD_START mode because the content of the memory can be saved during a power interrupt. The WARM_START and COLD_START status may be used in that case to inform the partition that data have been kept, and they may be reused when power recovers. The system integrator must inform the application developer the initial context differences between these two initialization modes.

COMMENTARY

> The **main process** is the only process that runs in COLD_START/WARM_START mode. This is the default process of the application. All other processes within the partition need to be created within the **main process** in order to declare their existence to the O/S. Once created, a process must be started in order for it to be executed. Any process started **during partition initialization** (i.e., to cause automatic execution following partition initialization) will not execute until the partition enters NORMAL_MODE (by calling the SET_PARTITION_MODE service). It is O/S implementation dependent whether the process used during **partition** initialization continues to run after setting OPERATING_MODE to NORMAL. From the viewpoint of portability, the application should not depend on the process continuing to run.

2.3.1.4.2 Partition Modes and Transitions

This section describes ARINC 653 partition modes and the characteristics expected when transitioning from one mode to another.

2.3.1.4.2.1 Initialization Mode Transition

Various events may cause a transition to the COLD_START or WARM_START modes: power interrupt, hardware reset, Health Monitoring action or the SET_PARTITION_MODE service.

For Entry Point management, the O/S will launch the partition at a unique entry point (the same Entry Point for COLD_START and WARM_START).

COMMENTARY

For some languages (Ada, C), an application's entry point is defined by the language's standard.

2.3.1.4.2.2 Transition Mode Description

(1a) COLD_START -> COLD_START

(1b) WARM_START -> WARM_START

The partition is in an initialization mode and restart is required to go back to the same initialization mode.

The parameter START_CONDITION returned by the GET_PARTITION_STATUS service **provides an application** the cause of entering the COLD_START or WARM_START mode. This parameter allows **the application associated with a** partition to manage its own processing environment on initialization error, depending on its own start context (e.g., create**/does not create a particular** process or port). It can use this information to prevent continuously restart of a partition due to persistent errors.

(2) WARM_START -> COLD_START

The partition is in an initialization mode and restart is required for another initialization mode.

The initial context of a WARM_START is considered to be more complete than an initial context of a COLD_START, so it is not possible to go from COLD_START to WARM_START directly or indirectly via a transition through the idle state. If the partition is in the COLD_START mode and the **defined HM recovery action is WARM_START, then a COLD_START is performed.**

(3a) COLD_START -> IDLE

(3b) WARM_START -> IDLE

(3c) NORMAL -> IDLE

When the partition calls the SET_PARTITION_MODE service with the OPERATING_MODE parameter set to IDLE, or when the Health Monitor makes the recovery action (i.e., according to the partition's health monitor table) to shutdown the partition.

(4a) COLD_START -> NORMAL

(4b) WARM_START -> NORMAL

When the partition has completed its initialization and calls the SET_PARTITION_MODE service with the OPERATING_MODE parameter set to NORMAL.

(5a) NORMAL -> COLD_START

(5b) NORMAL -> WARM_START

The partition is in the normal mode and restart is requested.

(6a) IDLE -> COLD_START

(6b) IDLE -> WARM_START

The only mechanism available to transition from the IDLE mode is an action external to the partition, such as power interrupt, **core** module reset, or application reset, if an external means exist.

2.3.2 Process Management

Within the ARINC 653 concept, a partition comprises one or more processes that combine dynamically to provide the functions associated with that partition. According to the characteristics associated with the partition, the constituent processes may operate concurrently in order to achieve their functional and real-time requirements. Multiple processes are therefore supported within the partition. **Processes within a partition share the same address space.**

An application requires certain scheduling capabilities from the operating system in order to accurately control the execution of its processes in a manner that satisfies the requirements of the application. Processes may be designed for periodic or aperiodic execution, the occurrence of a fault may require processes to be reinitialized or terminated, and a method to prevent rescheduling of processes is required in order to safely access resources that demand mutually-exclusive access.

A process is a programming unit contained within a partition which executes concurrently with other processes of the same partition. It comprises the executable program, data and stack areas, program counter, stack pointer, and other attributes such as priority and deadline. For references within this specification, the term “process” will be used in place of “task” to avoid confusion with the Ada task construct.

Access to process management functions is via the utilization of APEX services. The set of services called in the application software should be consistent with the certification/validation criticality level of the associated partition. Partition code executes in User mode only (i.e., no privileged instructions are allowed).

The partition should be responsible for the behavior of its **defined** processes. The processes are not **directly** visible outside of the partition.

2.3.2.1 Process Attribute Definition

In order for processes to **execute**, a set of unique attributes needs to be defined for each process. These attributes differentiate between the unique characteristics of each process as well as define resource allocation requirements.

A tabular description of the process attributes is given below. Fixed attributes are statically defined and cannot be changed once the partition has been **initialized**. Variable attributes are defined with initial values, but can be changed through the use of service requests once the **partition transitions to the NORMAL partition operating mode**.

FIXED ATTRIBUTES **(not included in the ARINC 653 XML-schema types)**

1. Name - Defines a value unique to each process in the partition.
2. Entry Point - Denotes the starting address of the process.
3. Stack Size - Identifies the overall size for the run-time stack of the process. **Each process is allocated a unique stack. Each process's stack may be configured to be identical or different in size.**
4. Base Priority - Denotes the **priority of the process defined when the process was started**.
5. Period - Identifies the period of activation for a periodic process. A distinct and unique value is used to designate a process as aperiodic.
6. Time Capacity - Defines the elapsed time within which the process should complete its execution.
7. Deadline - Specifies the type of deadline relating to the process, and may be "hard" or "soft". **When a process fails to meet its deadline, the O/S does not react differently between "hard" and "soft" deadlines. The O/S will take the action defined for deadline missed defined in the HM tables. If a process level error handler is defined, the process level error handler can utilize the GET_PROCESS_STATUS service to obtain the deadline type and take application specific actions based on "hard" versus "soft" deadlines.**
8. VARIABLE ATTRIBUTES

1. Current Priority - Defines the priority with which the process may access and receive resources. It is set to base priority at **process creation** and is dynamic at run-time.
2. Deadline Time - The deadline time is evaluated by the operating system to determine whether the process is satisfactorily completing its processing within the allotted time. **When a process is dormant (i.e., stopped), the deadline time returned by GET_PROCESS_STATUS is undefined.**
3. Process State - Identifies the current scheduling state of the process. The state of the process could be dormant, ready, running or waiting.

Once the process attribute information is defined for each process within the partition, a method is required to provide this information to the O/S. Normally, the linker provides a certain amount of attribute information (i.e., **entry point**, stack size, etc.) pertinent to the main (or initial) process of the system. Some linkers also allow for the creation of user-defined data and/or code segments during the link procedure.

2.3.2.2 Process Control

The system resources available to manage the processes of a partition are statically defined at build time and system initialization.

Processes are created (e.g., resources are allocated) and initialized during partition initialization. This implies that all the processes of a partition be defined in such a way that the necessary resource utilization for each process be determined at system build time. Each process is created only once during partition **initialization (i.e., on partition restart, the processes will be recreated)**.

A partition should be able to **restart (i.e.,** reinitialize) any of its processes at anytime and should also be able to prevent a process from becoming eligible to receive processor resources. Certain fault and failure conditions may necessitate a partition to restart or terminate any of its processes.

These conditions may arise due to either hardware or software faults, or as a result of a distinct operational phase of the application.

Each process has a priority level, and its behavior may be synchronous (periodic) or asynchronous **(aperiodic)**. Both types of processes can co-exist in the same partition. Any process could be preempted at any time by a **ready** process with a higher current priority. The process in the ready state with the highest (most positive) current priority is **selected for execution when** the partition is active.

The LOCK_PREEMPTION and UNLOCK_PREEMPTION services are included in the APEX specification to provide a means to control process preemption. Preemptibility control allows sections of code to be guaranteed to execute without preemption by other processes within the partition. If a process **that locked preemption** is interrupted by the end of a partition window, it is guaranteed to be the first to execute when the partition is resumed.

COMMENTARY

> Since Ada is a development language used for airline avionics, it is possible to use the Ada tasking construct to define each process. However, this is an undesirable approach since the tasking construct as defined by the language itself may not be sufficiently deterministic, nor specific enough, for the requirements of IMA. Therefore, the model specified herein defines a process with nearly identical characteristics of an Ada task without using the Ada tasking construct.
>
> Since Ada tasking is not explicit, services are provided to dynamically start and stop any process of a partition. The termination of a process may be defined in the IMA model as the situation which occurs when the executable code of a process runs to completion. Although it does not require any service, this situation has to be examined in order to fully describe the process behavior.
>
> There are four types of program units defined within the Ada language: Generic Units, Packages, Subprograms, and Tasks. Most processes will not lend themselves to generic units since each process will most likely have unique properties. The Task program unit should not be used to define processes due to the previously mentioned problem areas associated with using Ada tasking. The Package program unit, on the other hand, with a parameter-less procedure specification, is quite appropriate for the purpose of describing executable units in embedded systems. This provides the partition/process the ability to isolate and localize its internal data, types, procedures, and functions.

An O/S may or may not support static or dynamic creation for processes or any other communication mechanisms. This should be transparent to the application software.

The mechanisms used by the processes, for inter process communication and synchronization, are also created during the initialization phase, and are not destroyed.

To become active, a process not only needs to be created, but also needs to be started. At least one process in the partition is started shortly after creation. Active processes can start others, stop themselves or others, and restart as required by the application; this is governed by the process state transition model within the O/S. Processes are never destroyed (i.e., the memory areas assigned to the process are not de-allocated).

2.3.2.2.1 Process State Transitions

The O/S views the execution of a process in the form of a progression through a succession of states. Translation between these states is according to a defined process state transition model. Figure 2.3 depicts corresponding state transitions in accordance with the modes of the partition.

2.3.2.2.1.1 Process States

The process states as viewed by the O/S are as follows:

1. **Dormant** - A process that is ineligible to receive resources **for scheduling**. A process is in the dormant state before it is started and after it is terminated (or stopped).
2. **Ready** - A process that is eligible for scheduling. A process is in the ready state if it is able to be executed.
3. **Running** - The process that is currently executing on the processor. A process is in the running state if it is the current process in execution. Only one process can be executing **on a processor** at any time.
4. **Waiting** - A process that is not allowed to receive resources **eligible for scheduling** until a particular event **condition** occurs. A process is in the waiting state if one or both of the following are applicable:

   - The first reason is one of the following (waiting on a resource):
     - waiting on a delay
     - waiting on a semaphore
     - waiting on a period
     - waiting on an event
     - waiting on a message
     - waiting on the start of the partition’s NORMAL mode
   - The second reason is:
     - suspended (waiting for resume)

2.3.2.2.1.2 Process State Transitions Versus Partition Mode Transitions

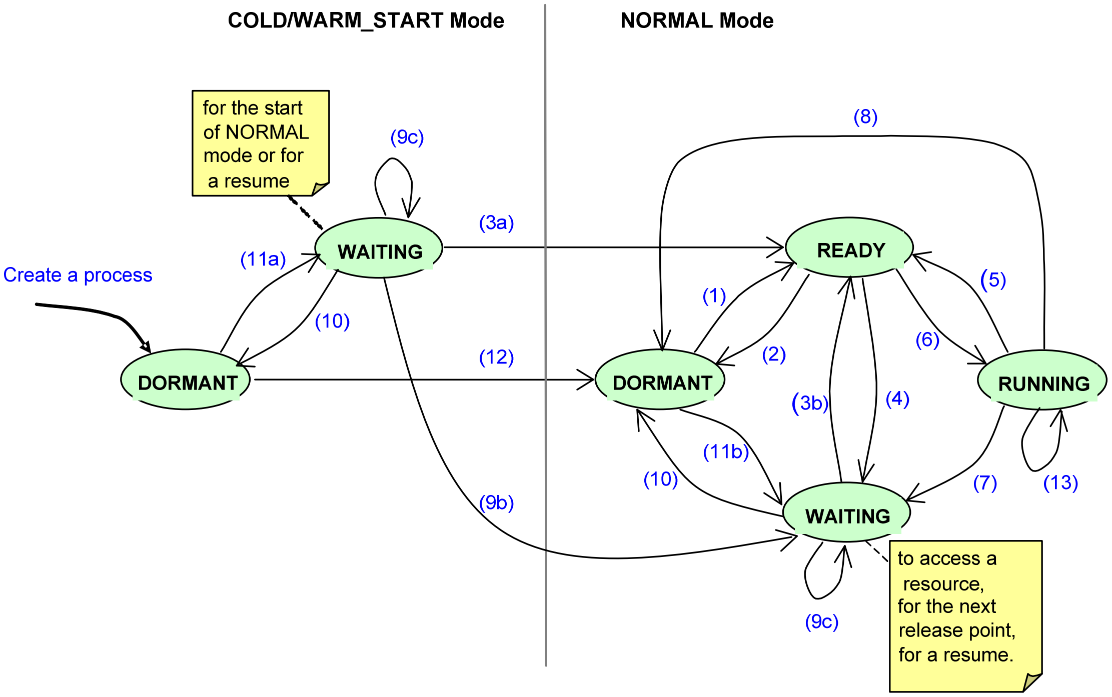


Figure 2.3 – Process States and State Transitions in Accordance with the Modes of the Partition

2.3.2.2.1.3 State Transitions

COLD_START and WARM_START modes correspond to the initialization phase of the partition.

State transitions occur as follows:

(1) Dormant - Ready

When the process is started by another process while the partition is in NORMAL mode.

(2) Ready - Dormant

When the process is stopped by another process while the partition is in NORMAL mode.

(3a) Waiting - Ready

When the partition transitions from the initialization phase to the NORMAL mode, and the process is an aperiodic process started during initialization phase (and was not suspended during initialization phase).

(3b) Waiting - Ready

Either when a suspended process is resumed, or the resource that the process was awaiting becomes available or the time-out expires.

Comment: A periodic process waiting on its period goes automatically to the ready state when the awaited release point is reached.

(4) Ready - Waiting

When the process is suspended by another process within the partition.

(5) Running - Ready

When the process waits on a DELAY_TIME of zero **and there are other ready processes that have the same priority value. Also when the process** is preempted by another process within the partition.

(6) Ready - Running

When the process is selected for execution.

(7) Running - Waiting

When the process suspends itself or when the process attempts to access a resource (semaphore, event, message, delay or period) which is not currently available and the process accepts to wait.

Comment: Waiting on the period means that the periodic process waits until the next release point by using the PERIODIC_WAIT service.

(8) Running - Dormant

When the process stops itself.

(9a) Waiting - Waiting

When a process already waiting to access a semaphore, an event, a message, or a delay is suspended. Also when a process which is both waiting to access a resource and is suspended, is either resumed, or the resource becomes available, or the time-out expires.

(9b) Waiting - Waiting

When the partition goes from Initialization phase to NORMAL mode, and the process is a periodic process that was started during Initialization phase, or the process is an aperiodic process that was suspended during Initialization phase, or an aperiodic process was delayed started during the Initialization phase.

(9c) Waiting - Waiting

When the process is suspended while the partition is in Initialization phase.

(10) Waiting - Dormant

When the process is stopped by another process within the partition. Note this applies to both initialization phase and NORMAL mode.

(11a) Dormant - Waiting

When the process is started and the partition is in Initialization phase.

(11b) Dormant - Waiting

When a periodic process is started (either with a delay or not) and the partition is in NORMAL mode.

(12) Dormant - Dormant

When the partition transitions from Initialization phase to NORMAL mode and the process has not yet been started.

(13) Running - Running

When the currently running process invokes an APEX service that generally results in a change in process state and it returns without changing the process sate (e.g., passes a DELAY_TIME of zero, or use of PERIODIC_WAIT under some conditions, see Section 2.3.3).

When the partition restarts in COLD_START or WARM_START mode then the previously created processes no longer exist and will need to be re-created.

State transitions may occur automatically as a result of certain APEX services called by the application software to perform its processing. They may also occur as a consequence of normal O/S processing, due to time-outs, faults, etc.

2.3.2.3 Process Scheduling

The main characteristics of the **process** scheduling model used at the partition level are:

1. The scheduling unit is an APEX process. One of the main activities of the O/S is to arbitrate the competition that results in a partition when several processes of the partition each want exclusive control over the processor (i.e., to control concurrency of operation).
2. Each process has a **current** priority.
3. The scheduling algorithm is priority preemptive. The O/S manages the execution environment for the partition, dispatching and preempting processes based on their respective priority levels and current states. **If the current process is higher priority than any process in the ready state, the current process will not be preempted.** During any process rescheduling event (caused either by a direct request from that process or any partition internal event), the O/S always selects the highest priority process in the ready state within the partition to receive processor resources. **If several processes have the same current priority, the O/S selects the process that has been in the ready state the longest time at that priority value.**

   Note however that if process A is running and is preempted by higher-priority process B, process A (given that its priority was not modified) will be selected to run before other processes that have the same priority as process A.

   If process A lowers its priority to be the same priority as ready process C (and preemption is not locked), the current process A will be the newest process ready at that priority level and will be preempted by process C.

   The selected process will control processor resources until another process reschedule event occurs.
4. Periodic and aperiodic scheduling of processes are both supported.

COMMENTARY

The requirement for periodic and aperiodic scheduling of processes does not imply that the O/S must distinguish between two types of processes. It **means** that the APEX interface provides the application software with **the ability** to request scheduling of processes on a periodic or aperiodic (e.g., event-driven) basis.

5. All the processes within a partition share the resources allocated to the partition.

When periodic processes are started, their start-time is relative to the partition periodic processing start defined for the partition in the configuration tables. Each of the partition's allocated partition windows are individually designated as whether they are or are not partition periodic processing starts. For periodic processes, the first release point of the process is relative to the start of next partition window defined as a periodic processing start. For periodic processes, subsequent process release points will occur at the process's defined period.

Aperiodic processes only have an initial release point. When aperiodic processes are started, their start time is relative to the current time.

2.3.2.4 Process Priority

Two types of priority are represented in the system, current and base. Current priority is the priority used by the O/S for scheduling and process queues. Base priority is the priority used when the process is started.

2.3.2.5 Process Waiting Queue

When a process makes a service request that causes the process to block, the process's state is set to WAITING, and the process is placed on a process queue. Associated with each potentially blocking resource (e.g., queuing port, buffer, semaphore, event) is a process queue. Processes are placed on the process queue based on the queuing discipline (FIFO, priority) defined when the create service for the resource was invoked and the current priority of the process (when queuing discipline is priority order).

2.3.2.6 Process Preemption Locking

Support for process preemption locking is provided. Using preemption locking, normal process rescheduling operations of the operating system are prevented while its processes are accessing critical sections or resources shared by multiple processes of the same partition. A critical section may be access for specific areas of memory, certain physical devices, or simply the normal calculations and activity of a particular process.

The LOCK_PREEMPTION and UNLOCK _PREEMPTION services are used to control the partition's lock level. When the partition's lock level value is greater than 0, preemption is disabled and the partition is in the locked condition (i.e., no process scheduling occurs). When the value is 0, preemption is enabled and the partition is in the unlocked condition (i.e., process scheduling can occur).

Preemption locking has no impact on the scheduling of other partitions. It only affects scheduling of processes within a partition.

COMMENTARY

The ability to intervene with the normal rescheduling operations of the operating system does not imply that the application software is directly controlling the O/S software. Since lock preemption is provided by the O/S and all resultant actions and effects are known beforehand, the integrity of the O/S remains intact and unaffected by the service requests.

2.3.3 Time Management

Time management is an important characteristic of an O/S used in real-time systems. Time is unique and independent of partition execution within a module. All time values or capacities are related to this unique time and are not relative to any partition execution. The O/S provides time slicing for partition scheduling, deadline, periodicity, and delays for process scheduling, and time-outs for intrapartition and interpartition communication in order to manage time. These mechanisms are defined through attributes or services.

A time capacity is associated with each process. It represents the response time given to the process for satisfying its processing requirements.

COMMENTARY

Time capacity is associated with periodic and aperiodic processes and is used to set process deadline. The deadline associated with a blocked process will continue to be evaluated while it is blocked. The deadline associated with periodic processes is reset by invoking the PERIODIC_WAIT service. The deadline associated with aperiodic processes is reset by using the stop and start services.

When a process is started (either explicitly via a service request or at the end of the initialization phase), its deadline is set to the value of current time plus time capacity. This deadline time may be postponed by mean of the REPLENISH service. Note that this capacity is an absolute duration of time, not an execution time. This means that a deadline overrun will occur even when the process is not running inside or outside the partition window, but will be acted upon only inside a partition window of its own partition.

COMMENTARY

Since the deadline time is absolute, modifications to the partition’s time windows can result in deadline overruns. Dependencies between process deadline and the partition time windows include whether a partition’s duration is divided into multiple time windows.

Time replenishment determines the next deadline value as represented in **Figure 2.3.3.**

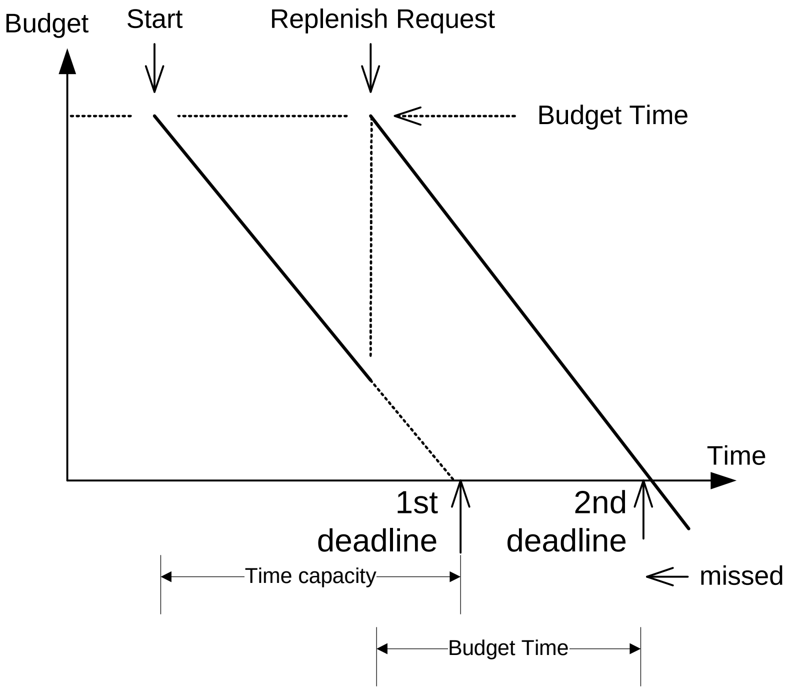


Figure 2.3.3 – Time Replenishment

As long as the process performs its entire processing without using its whole time capacity, the deadline is met. If its processing requires more than the time capacity, the deadline is missed. **When the deadline is missed, an error will be raised to the HM functionality (see Section 2.4).**

COMMENTARY

If the HM functionality does not resolve the deadline miss error (i.e., error handler did not modify the erroneous process' state, or an "ignore" recovery action was utilized), the subsequent behavior of the process that missed the deadline is implementation dependent. It is strongly recommended (for portability reasons) that deadline miss errors be managed by health monitoring instead of relying on implementation dependent behavior. Implementation dependent behaviors could include:

- **The process retains its pre-deadline miss state (e.g., still running if it was the previously running process), with a new deadline time not being set for the process until the process invokes the PERIODIC_WAIT service (i.e., expiration of a process' deadline does not affect the process' period). Invocation of PERIODIC_WAIT during the process' next period results in the process being eligible to run immediately (i.e., to perform its intended function for its next period). Failure to invoke a 2nd PERIODIC_WAIT before the deadline associated with the process' next period results in another deadline miss HM event.**
- **The process retains its pre-deadline miss state (e.g., still running if it was the previously running process), with the core software extending the process' deadline time to its next release point (i.e., skip one process period and utilize it to complete the overrun of the previous period). Invocation of PERIODIC_WAIT during the extended period results in process waiting on its next release point. Failure to invoke**

PERIODIC_WAIT during the extended period results in another deadline miss HM event.

A time-out delay or deadline can expire outside the partition window. It is acted upon at the beginning of the next partition window.

COMMENTARY

> The notion of time used here is local to a module and is needed for time management within the partitions on that module. If a partition needs a timestamp for a reason such as fault reporting, a different time may be used. That time value could come from an aircraft clock via normal interpartition communication means.

2.3.4 Memory Management

Partitions, and therefore their associated memory spaces, are defined during system configuration **and initialization**. There are no memory allocation services in the APEX interface.

2.3.5 Interpartition Communication

A major part of this standard is the definition of the communication between APEX partitions.

Interpartition communication is a generic expression used in this standard. **The interpartition communication definitions contained in this standard are intended to facilitate communications between ARINC 653 application partitions residing on the same module or on different modules, as well as communications between an ARINC 653 application partition and** non-ARINC 653 equipment external to **that partition's host** core module. Standard interpartition communication is a basic requirement for support of reusable and portable application software.

All interpartition communication is conducted via messages. A message is defined as a contiguous block of data of finite length. A message is sent from a single source to one or more destinations. The destination of a message is a partition, and not a process within a partition.

COMMENTARY

> The expression "contiguous block" means a sequential data arrangement in the source and destinations memory areas. Sending a message without format conversion is achieved by copying the message from memory to memory via the communication network. This principle does not require the message to be entirely transmitted in a single packet.

The APEX interface supports communication of the full message, regardless of any message segmentation applied by the O/S. Note that this does not prohibit a partition from decomposing a large message into a set of smaller ones and communicating them individually. The O/S regards them purely as separate unrelated messages.

COMMENTARY

> From the application viewpoint, a message is a collection of data which is either sent or received to/from a specific port. Depending on the **port implementation**, the messages allowed to be passed in that port may be either fixed or variable length.

The basic mechanism for linking partitions by messages is the channel. A channel defines a logical link between one source and one or more destinations, where the source and the destinations may be one or more partitions. It also specifies the mode of transfer of messages from the source to the destinations together with the characteristics of the messages that are to be sent from that source to those destinations.

Partitions have access to channels via defined access points called ports. A channel consists of one or more ports and the associated resources. A port provides the required resources that allow a specific partition to either send or receive messages in a specific channel. A partition is allowed to exchange messages through multiple channels via their respective source and destination ports. The channel describes a route connecting one sending port to one or several receiving ports.

From the perspective of a partition, APEX communication services support the transmission and receipt of messages via ports, and are the same regardless of the system boundaries crossed by the communicated message. Therefore, the system integrator (not the application developer) configures the channel connections within a module and the channel connections between a module and components external to the module.

2.3.5.1 Communication Principles

ARINC 653 communications are based on the principle of transport mechanism independence at partition level. It is assumed that the underlying transport mechanism transmits the messages and ensures that the messages leave the source port and reach the destination ports in the same order. Communications between partitions, or between partitions and external entities, use the same services and are independent of the underlying transport mechanism. Messages exchanged between two compliant ARINC 653 applications are identical regardless of whether the applications reside on the same module, or on different modules.

The **core module** is responsible for encapsulating and transporting messages, such that the message arrives at the destination(s) unchanged. Any fragmentation, segmentation, sequencing, and routing of the message data by the **core module**, required to transport the data from source to destination, is invisible to the application(s). The **core module** is responsible for ensuring the integrity of the message data, i.e., messages should not be corrupted in transmission.

At the application level, messages are atomic entities (i.e., either the whole message is received or nothing is received). Applications are responsible for assuring the data meets requirements for processing by that application. This might include range checks, voting between different sources, or other means.

It is the responsibility of the application designer, in conjunction with the system integrator, to ensure that the transport mechanism chosen meets the message transport latency and reliability required by the application.

COMMENTARY

This technique provides:

- The system integrator with freedom to optimize the allocation of partitions to processing resources within a core module or among multiple core modules.
- Portability of applications across platforms.
- Network technology upgrades without significant impact to applications.

Communication within a partition use services that are synonymous to the external communication services.

The following are limitations on the behavior of APEX communications:

1. The performance and integrity of interpartition communications is dependent on the underlying transport mechanism (i.e., communication media and protocols), which is beyond the scope of this standard. When defining the functional behavior and integrity of queuing and sampling port services, it is assumed the underlying transport mechanism supports this behavior. It is the systems integrator responsibility to specify the end-to-end behavior of the communications system, assure functional requirements are met, and communicate behavioral differences between this standard and the implementation to application providers.
2. Although data abstraction should assure parameter consistency, equivalent performance between differing transport mechanisms is not guaranteed.
3. Synchronous behavior between communication ports is not guaranteed by the APEX interface. This may or may not be a feature of the underlying system.
4. The **O/S** should ensure that the messages provided by the application are transmitted to the transport mechanism in the same order. The **O/S** should ensure that the messages received from the transport mechanism are delivered to the application in the same order. **It is assumed that the transport mechanism maintains ordinal integrity of messages between source and destination.**
5. Any particular message can only originate from a single source.
6. When a recipient accesses a new instance of a message, it can no longer request access to an earlier instance. It cannot be assumed that the O/S will save old versions of messages.

2.3.5.2 Message Communication Levels

Messages may be communicated across different message passing boundaries as defined below:

1. Within core modules - allows messages to be passed between partitions supported by the same core module. Whether the message is passed directly, or across a data bus, is configurable by the system integrator for each message source.
2. Between the core modules - allows messages to be passed between multiple core modules via a communication bus.
3. Between core modules and a non-ARINC 653 component - allows messages to be passed between core modules and devices that do not host an ARINC 653 O/S (traditional LRUs, sensors, etc.) via various communication buses.

In traditional LRUs which use an APEX interface, messages may be passed either directly between partitions on the same LRU (for LRUs supporting multiple partitions) or between partitions and the LRU buses.

Interpartition communication conducted externally across core module boundaries should conform to the appropriate message protocol. The message protocol to be used for this communication is system specific. The system partition and/or the O/S I/O device driver are responsible for communicating these messages over the communication bus, taking the physical constraints of the bus into consideration.

Characteristics of communication media may require the segmentation of an externally communicated message. This segmentation is transparent to the application software and is the responsibility of the I/O mechanisms provided by the O/S.

COMMENTARY

> This version of ARINC 653 does not address the communication protocol between core modules. This issue may be addressed in a future version of this standard.

2.3.5.3 Message Types

The messages may be of the types described in the following sections.

2.3.5.3.1 Fixed/Variable Length

A fixed length message has a defined constant size for every occurrence of the message. A variable length message can vary in size, and therefore, requires the source to specify the size within the message when transmitting it.

COMMENTARY

> Fixed length is best suited to the transmission of measurements, commands, status, etc. Variable length is more appropriate to the transmission of messages whose amount of data varies during run time (e.g., list of airframes for the TCAS function).

2.3.5.3.2 Periodic/Aperiodic

Periodic means that the communication of a particular message is performed on a regular iterative basis. Aperiodic means that the communication is not necessarily periodic.

COMMENTARY

> For periodic messages, it is usual for recipients to be designed to cope with intermittent loss of data. Typically, the last valid data sample is used by the recipient until either the continued data loss is unacceptable or a new valid sample is received.
>
> Periodic messages are best suited to the communication of continuously varying data (e.g., Air Speed, Total Pressure). Aperiodic messages are best suited to the communication of irregular events that may occur at random intervals (e.g., due to an operator-instigated action).
>
> No distinction is made by the O/S between periodic and aperiodic messages as the instances of message generation are implicitly defined according to the nature of the partition’s runtime activation (i.e., the periodicity of a message is dictated by the periodicity at which the message is sent).
>
> Discrete events should not be reported in single periodic messages or should be acknowledged within applications, at a level above that of the message passing protocol. Discrete events are best suited to being announced in aperiodic messages.

2.3.5.3.3 Broadcast, Multicast and Unicast Messages

This standard provides support for broadcast, multicast and unicast messages. A broadcast message is sent from a single source to all destinations. A multicast message is sent from a single source to more than one destination. A unicast message is sent from a single source to a single destination. For queuing mode, only unicast messages are required.

COMMENTARY

The behavior of broadcast and multicast queuing is implementation dependent.

This standard does not directly support client/server messages. Client/server support may be implemented by applications on top of queuing or sampling ports.

2.3.5.4 Message Type Combinations

(This section deleted by Supplement 1.)

2.3.5.5 Channels and Ports

Channels and ports are defined as part of the system configuration activities. The definition of channels is not confined to the core module O/S but is rather distributed on the constituent **core** modules and LRUs of the system. Each communication node (core module, gateway, I/O module, etc) is assumed to be separately configurable (i.e., through configuration tables) to define how messages are handled at that node. It is the system integrator responsibility to ensure that the different nodes crossed by each channel are consistently configured. The consequence is that the source, destinations, mode of transfer, and unique characteristics of each channel cannot be changed at run-time.

2.3.5.6 Modes of Transfer

Each individual **port** may be configured to operate in a specific mode. Two modes of transfer may be used: sampling mode and queuing mode.

COMMENTARY

It may be possible to configure communication such that ports operating in different modes (sampling or queuing) can be connected to the same channel; however, this is dependent on the underlying media and is outside the scope of this standard.

A full set of services has to be provided to the application software to accommodate the two modes of transfer, sampling mode and queuing mode, and the two transfer directions, source and destination. Only the subset of services whose functions are compatible with the port configuration (mode, transfer direction) will execute correctly when that port is addressed; others will return an appropriate value of the return code indicating that the request failed.

Sampling ports and queuing ports can be used in any mode, i.e., NORMAL, COLD_START and WARM_START. It is assumed that the underlying system supports the behavior defined herein.

Messages are allowed to be segmented/reassembled. The segmentation and reassembling of messages are transparent to the application.

2.3.5.6.1 Sampling Mode

In the sampling mode, successive messages typically carry identical but updated data. No queuing is performed in this mode. A message remains in the source port until it is transmitted by the channel or it is overwritten by a new occurrence of the message, whichever occurs first. This allows the source partition to send messages at any time. Each new instance of a message overwrites the current message when it reaches a destination port, and remains there until it is overwritten. This allows the destination partitions to access the latest message.

COMMENTARY

> The periodicity of transmission is determined by the source application in its sending of the messages. Reception of such messages is additionally influenced by the latency in communicating the messages through the intermediate communication media (i.e., backplane bus, gateway, LRU specific media).

It is the responsibility of the underlying port implementation to properly handle data segmentation. A partial message will not be delivered to the destination port.

2.3.5.6.2 Queuing Mode

In the queuing mode, each new instance of a message may carry uniquely different data and is not allowed to overwrite previous ones during the transfer. No message should be unintentionally lost in the queuing mode.

The ports of a channel operating in queuing mode are allowed to buffer multiple messages in message queues. A message sent by the source partition is stored in the message queue of the source port until it is transmitted by the channel. When the message reaches the destination port, it is stored in a message queue until it is received by the destination partition. A specific protocol is used to manage the message queues and transmit messages in the source port and in the destination port in a first-in/first-out (FIFO) order (see Section 2.3.5.1, Communication Principles item d). This allows the source and destination ports to manage the situation where either the source or destination queues are full. Application software is responsible for handling overflow when the queuing port is full.

Variable length messages are supported in the queuing mode.

COMMENTARY

> Since the amount of data carried by a message may vary considerably, encapsulating the data in a fixed length message may not be acceptable because of excessive usage of RAM and bus bandwidth. Applications are therefore expected to manage different types of messages, each with an appropriate length. The channel does not need to distinguish between those different types of messages, but is only required to accept transmitting successive messages with different lengths.

It is not a requirement to have true aperiodic transmission of messages between the source and destination ports.

A partial message will not be delivered to the destination port in the queuing mode.

2.3.5.7 Port Attributes

In order for the required communication resources to be provided to the partitions, a set of unique attributes needs to be defined for each port. These attributes enable the O/S to control and maintain the operation of ports and portions of channels located within the core module (or the LRUs). **The port attributes are defined in the configuration tables or by the application when invoking a service call to create the port. The required port attributes are described in the following sections.**

2.3.5.7.1 Partition Identifier

The partition identifier denotes the partition which is allowed to have access to the port.

2.3.5.7.2 Port Name

The port name attribute consists of a pattern that uniquely identifies the port within the partition that has access to it. This name is intended to be used by the application software to designate the port.

COMMENTARY

> By using port names instead of addressing directly the source/destination partitions, the application software is more independent of the communication network architecture**. The configuration of port names is the responsibility of the** system integrator. It should be noted that a well-chosen port name should refer to the data passed in the port rather than to the producer/consumers of those data ***(***e.g., “MEASURED ELEVATOR POSITION”).

2.3.5.7.3 Mode of Transfer

The mode of transfer attribute denotes the mode (i.e., sampling mode or queuing mode), which is expected to be used to manage the messages in the port.

2.3.5.7.4 Direction

The direction attribute indicates whether the port allows messages to be sent (the partition is viewed as a source) or received (the partition is viewed as a destination).

2.3.5.7.5 Maximum Message Size

The maximum message size attribute defines the maximum number of bytes a single message may contain for the port.

2.3.5.7.6 Maximum Number of Messages

The maximum number of messages attribute applies to messages in the queuing mode only. This attribute defines the maximum number of messages the queuing port must handle without loss.

COMMENTARY

> The **maximum message size and maximum number of messages** attributes are used to define storage areas in the memory space of the partition, allowing messages passed in the port to be temporarily buffered. Depending on the mode of transfer and the transfer direction, different message storage areas may be required:
>
> 1. Sampling mode, send direction: No particular message storage is required when a send request is issued by the application software.
> 2. Sampling mode, receive direction: A one-message area is required to buffer the last correct message received by the partition.
> 3. Queuing mode, send direction: A message queue, managed on a FIFO basis, is required to buffer the successive messages to be sent. The queue is filled upon send requests issued by the application software, and cleared as the O/S moves the messages from the queue.
> 4. Queuing mode, receive direction: A message queue, managed on a FIFO basis, is required to buffer the successive correct messages received by the partition. The queue is filled as the O/S moves correct received messages to the queue, and cleared upon receive requests issued by the application software.

COMMENTARY

In all cases, applications should allocate enough memory space to assure the configuration defined maximum message size can be received, regardless of any application protocol defined size assumptions.

2.3.5.7.7 Refresh Period

The refresh **period** attribute applies to received messages in the sampling mode only. This allows the O/S to control whether correct messages arrive at a **specified** rate in the port, regardless of the receive request rate.

2.3.5.7.8 Port Mapping

The **port** mapping attributes define the connection between that port and the physical communication media (e.g., memory, backplane bus,)**. The port mapping consists of attributes defining a physical address and/or procedure that may be used to map the port. Use, meaning, and limitations of these attributes are implementation dependent.**

COMMENTARY

Messages received by a port may originate either from another port of the **same module or via a inter-module communications mechanism.** Messages sent by a port are routed to one or more other ports of the module and/or the **inter-module communications mechanism** in the sampling mode, whereas they are routed to either one other port **of the module or the inter-module communications mechanism** in the queuing mode.

Ports can be mapped to physical addresses or to procedures (e.g., device drivers), both of which can map the ports to a backplane or communications device. A procedure mapping might be used when an intelligent I/O processor is not available to move data to/from an external device. In addition, the O/S implementer may define any other suitable mapping mechanisms. Any mechanism for mapping ports must insure that multiple partitions do not have write access to the address space to which the port is being mapped whether it is accomplished by a physical address mapping, a procedure mapping, or some other mechanism. The notion of shared memory can only be supported when the **O/S** controls the access to the memory location.

2.3.5.8 Port Control

The core module resources required to manage a port are statically defined at build time and initialized after power is applied to the core module. Once the ports assigned to a partition have been initialized, the application software is allowed to perform send or receive operations in those ports. The O/S does not ensure that the channels connected to the ports have been entirely initialized. The reason is that the channels may cross different nodes, and these nodes **may** not necessarily be initialized simultaneously (i.e., if located on different cabinets).

Creating a port associates a port identifier with a port name. The port identifier is an O/S-defined value associated with the port name at the core module initialization. Use of the port identifier allows more efficient access to the port resources than use of the port name.

COMMENTARY

> If messages are transmitted in a channel and some portions of that channel have not been initialized, then a fault will be detected either by the destination ports (message freshness check) in the sampling mode, or by the source port (**lack of response**) in the queuing mode.

Either fixed or variable length messages are allowed to be communicated via a port. A length indication is provided by the application software when sending a message and by the O/S when the application software receives a message.

Depending on its mode of transfer and its transfer direction, a port operates in different ways:

1. Sampling mode, send direction: Each new message passed by an application's send request overwrites the previous one. Each occurrence of a message cannot be transmitted more than once.
2. Sampling mode, receive direction: Each correct (internally consistent) new received message is copied to the temporary storage area of the port where it overwrites the previous one. This area is allowed to be polled at random times, upon application software receive requests. The copied message and a validity indicator are returned to the application software. The validity indicator indicates whether the age of the copied message is consistent with the required refresh rate defined for the port. The age of the copied message is defined as the difference between the value of the system clock (when READ_SAMPLING_MESSAGE is called) and the value of the system clock when the OS copied the messages from the channel into the destination port.
3. Queuing mode, send direction: Each new message passed by an application's send request is temporarily stored in the send message queue of the port. If the queue is full or contains insufficient space for the entire message to be stored, then the requesting process of the partition enters the waiting state or the send request may be cancelled. The queued messages are segmented **as required to satisfy the underlying communication media** before they are transmitted in FIFO order.
4. Queuing mode, receive direction: Each correct (internally consistent) new received message is moved to the receive message queue. The oldest message in the message queue is removed from the queue and passed to the application software upon receipt of the receive request. If the message queue is empty, the requesting process may enter the waiting state or the receive request may be cancelled. The channel protocol may prevent further segments from being received when the receive message queue is full, and request resending of failed segments. This applies only to unicast transmissions.

2.3.5.9 Process Queuing Discipline

The communication between partitions is done by processes which are sending or receiving messages. In queuing mode, processes may wait on a full message queue (sending direction) or on an empty message queue (receiving direction). Rescheduling of processes will occur when a process attempts to wait. The use of time-outs will limit or avoid waiting times.

Processes waiting for a port in queuing mode are queued in FIFO or priority order. In the case of priority order, for the same priority, processes are also queued in FIFO order. The queuing discipline is defined **by the application as part of the service call used for** creation of the port.

COMMENTARY

> When queued messages requiring responses are passed between partitions with a non-zero wait time, care must be used to ensure that faults (such as excessive delays or missing data from the other partition) do not result in undesired effects upon the receiving partition.

2.3.6 Intrapartition Communication

Provisions for processes within a partition to communicate with each other without the overhead of the global message passing system are included as part of this standard. The intrapartition communication mechanisms are buffers, blackboards, semaphores, and events. Buffers and blackboards provide general inter-process communication (and synchronization), whereas semaphores and events are provided for inter-process synchronization. All intrapartition message passing mechanisms must ensure atomic message access (i.e., partially written messages cannot be read).

There are no configuration table attributes associated with intrapartition communications. Attributes associated with intrapartition communications are specified by the application when invoking the corresponding create operation.

As with process and port creation, the amount of memory required to create intrapartition communication mechanisms is allocated from the partition's memory resources, which is defined at system build time. Processes can create as many intrapartition communication mechanisms as the partition's memory resources will support.

2.3.6.1 Buffers and Blackboards

Inter-process communication of messages is conducted via buffers and blackboards which can support the communication of messages between multiple source and destination processes. The communication is indirect in that participating processes address the buffer or blackboard rather than the opposing processes directly, thus providing a level of process independence. Buffers allow the queuing of messages whereas blackboards do not allow message queuing.

Rescheduling of processes will occur when a process must wait for a message. The use of time-outs will limit or avoid waiting times.

2.3.6.1.1 Buffers

In a buffer, each new instance of a message may carry uniquely different data and therefore is not allowed to overwrite previous ones during the transfer.

Buffers are allowed to store multiple messages in message queues. A message sent by the sending process is stored in the message queue in first-in/first-out (FIFO) order. **When using buffers, no messages are lost (the sender will block).** The number of messages that can be stored in a buffer is determined by the **defined maximum number of messages**, **which** is specified at creation time.

The CREATE_BUFFER operation creates a buffer for use by any of the processes in the partition. At creation, the buffer's maximum message size, maximum number of messages, and queuing discipline are defined.

Processes waiting on a buffer are queued in FIFO or priority order. In the case of priority order, processes with the same priority are queued in FIFO order. The queuing discipline is defined at the creation of the buffer.

If there are processes waiting on a buffer and if the buffer is not empty, the queuing discipline algorithm (FIFO or priority order) will be applied to determine which queued process will receive the message. The O/S will remove this process from the process queue and will put it in the ready state. The O/S will also remove the message from the buffer message queue.

Rescheduling of processes will occur when a process attempts to receive a message from an empty buffer or to send a message to a full buffer. The calling process will be queued for a specified amount of real-time. If a message is not received or is not sent in that amount of time, the O/S will automatically remove the process from the queue and put it in the ready state.

2.3.6.1.2 Blackboards

For a blackboard, no message queuing is allowed. Any message written to a blackboard remains there until the message is either cleared or overwritten by a new instance of the message. This allows sending processes to display messages at any time, and receiving processes to access the latest message at any time.

The CREATE_BLACKBOARD operation creates a blackboard for use by any of the process in the partition. At creation, the blackboard’s maximum message size is defined.

A process can read a message from a blackboard, display a message on a blackboard, or clear a blackboard.

Rescheduling of processes will occur when a process attempts to read a message from an empty blackboard. The calling process will be queued for a specified amount of real-time. If a message is not displayed in that amount of time, the O/S will automatically remove the process from the queue and put it in the ready state.

When a message is displayed on the blackboard, the O/S will remove all waiting processes from the process queue and will put them in the ready state. The message remains on the blackboard.

When a blackboard is cleared, it becomes empty.

2.3.6.2 Semaphore and Events

Inter-process synchronization is conducted using semaphores and events. Semaphores provide controlled access to resources; events support control flow between processes by notifying the occurrences of conditions to waiting processes.

2.3.6.2.1 Semaphores

The semaphores defined in this standard are counting semaphores, and are commonly used to provide controlled access to partition resources. A process waits on a semaphore to gain access to the **partition** resource, and then signals the semaphore when it is finished. A semaphore’s value in this example indicates the number of currently available **partition** resources.

The CREATE_SEMAPHORE operation creates a semaphore for use by any of the process in the partition. At creation, the semaphore’s initial value, maximum value, and queuing discipline are defined.

Processes waiting on a semaphore are queued in FIFO or Priority order. In the case of Priority order, for the same priority, processes are also queued in FIFO order **(with the oldest waiting process being at the front of the FIFO)**. The queuing discipline is defined at the creation of the semaphore.

The WAIT_SEMAPHORE operation decrements the semaphore's value if the semaphore is not already at its minimum value of zero. If the value is already zero, the calling process may optionally be queued until either a signal is received or until a specified amount of real-time expires.

The SIGNAL_SEMAPHORE operation increments the semaphores value. If there are processes waiting on the semaphore, the queuing discipline algorithm, FIFO or priority order, will be applied to determine which queue process will receive the signal. Signaling a semaphore where processes are waiting increments and decrements the semaphores value in one request. The end result is there are still no available resources, and the semaphore value remains zero.

Rescheduling of processes will occur when a process attempts to wait on a zero value semaphore, and when a semaphore is signaled that has processes queued on it.

When a process is removed from the queue, either by the receipt of a signal or by the expiration of the specified time-out period, the process will be moved into the ready state unless another process has suspended it.

2.3.6.2.2 Events

An event is a communication mechanism which permits notification of an occurrence of a condition to processes which may wait for it. An event is composed of a bi-valued state variable (states called "up" and "down") and a set of waiting processes (initially empty).

COMMENTARY

> On the arrival of an aperiodic message, a receiving process may send a synchronous signal (event) to another process.

Processes within the same partition can SET and RESET events and also WAIT on events that are created in the same partition. As with process and message buffer creation, the amount of memory space required to create events is allocated from the partition's memory which is defined at system build time.

The CREATE_EVENT operation creates an event object for use by any of the process in the partition. Upon creation the event is set in the state "down."

To indicate occurrence of the event condition, the SET_EVENT operation is called to set the specified event in the up state. All of the processes waiting on that event are then moved from the waiting state to the ready state, and a scheduling takes place.

The resume for waiting processes should appear to be atomic, so that all processes are available for scheduling at the same time. The order of execution should only depend on the intra-partition process scheduling policy.

The RESET_EVENT operation sets the specified event in the state "down."

The WAIT_EVENT operation moves the current process from the running state to the waiting state if the specified event is "down" and if there is a specified time-out that is greater than 0. Scheduling takes place.

The process goes on executing if the specified event is “up” or if there is a conditional wait (event down and time-out is less than or equal to 0).

Rescheduling of processes will occur when a process attempts to wait on an event which is “down” (reset before by another process or during initialization) and when a process sets an event “up”.

The GET_EVENT_STATUS operation returns the count of waiting processes and the state of the specified event.

2.4 Health Monitor

In general, Health Monitor (HM) functions are responsible for responding to and reporting hardware, application and O/S software faults and failures. ARINC 653 supports HM by providing HM configuration tables and an application level error handler process.

The HM helps to isolate faults and to prevent failures from propagating. Components of the HM are contained within the following software elements:

- O/S – All HM implementations will use the ARINC 653 O/S. The O/S uses the HM configuration tables to respond to pre-defined faults.
- Application Partitions – Where faults are system specific and determined from logic or calculation, application partitions may be used, passing fault data to the O/S via the HM service calls or to an appropriate system partition.
- System Partitions – System partitions can be used by the platform integrators **for error management.**

Figure 2.4 illustrates the ARINC 653 HM decision logic. The decision logic is further described in the following subsections and Section 2.5.2.3.

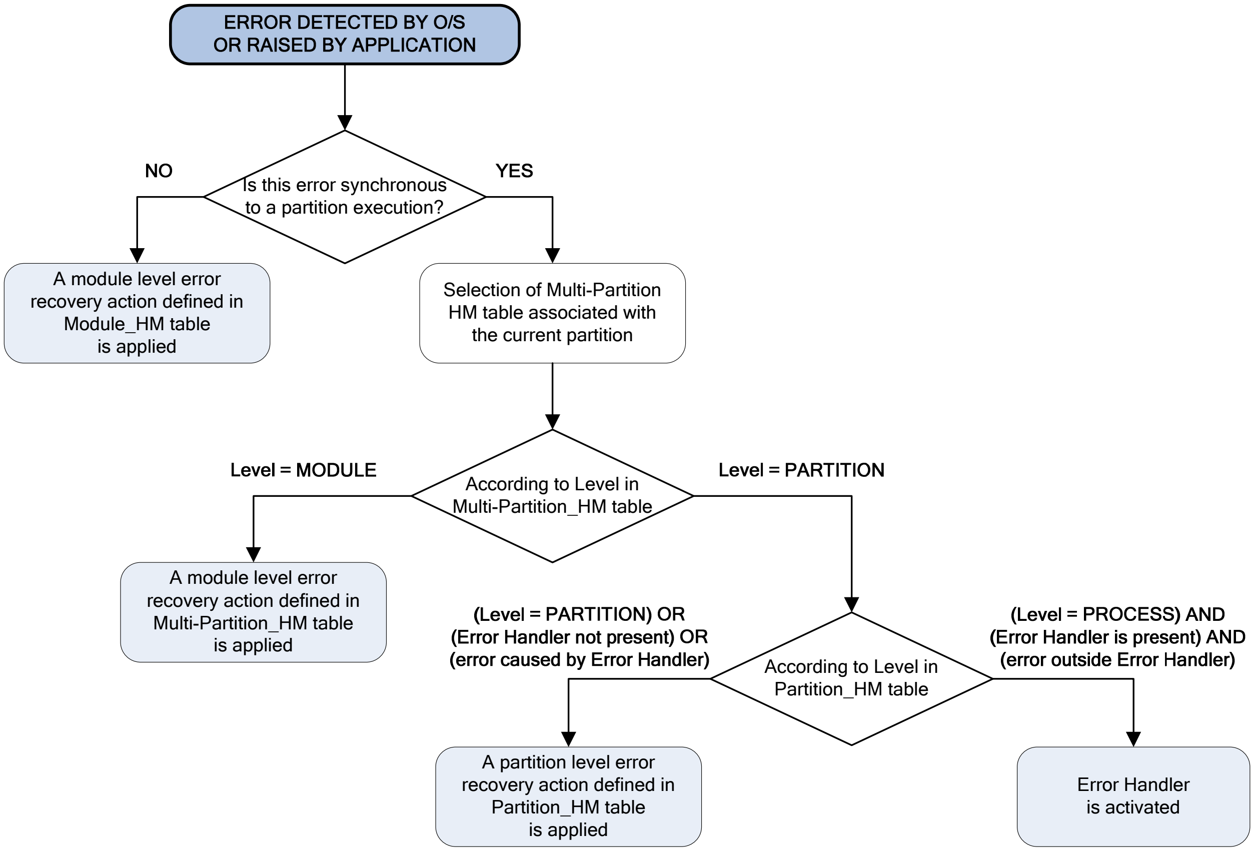

ERROR DETECTED BY O/S
OR RAISED BY APPLICATION
Is this error synchronous to a partition execution?
NO
YES
A module level error recovery action defined in Module_HM table is applied
Selection of Multi-Partition HM table associated with the current partition
According to Level in Multi-Partition_HM table
Level = MODULE
Level = PARTITION
A module level error recovery action defined in Multi-Partition_HM table is applied
According to Level in Partition_HM table
(Level = PARTITION) OR (Error Handler not present) OR (error caused by Error Handler)
(Level = PROCESS) AND (Error Handler is present) AND (error outside Error Handler)
A partition level error recovery action defined in Partition_HM table is applied
Error Handler is activated

Figure 2.4 – HM Decision Logic

2.4.1 Error Levels

Errors may occur at the module, partition, or process level. These error levels are described in the following sections. The level of an error is defined in the **module HM, multi-partition HM and partition HM tables** in accordance with the detected error and the state of the system.

COMMENTARY

> System-level errors and their reporting mechanisms are outside the scope of this document. It is the responsibility of the system integrator to ensure the system-level error handling and lower-level error handling are consistent, complete and integrated.

2.4.1.1 Process Level Errors

A process level error impacts one or more processes in a partition or the entire partition. Examples of process level errors are:

- Application error raised by an application process
- Illegal O/S request
- Process execution errors (overflow, memory violation, etc.)

The HM will not violate partitioning when handling process level errors

2.4.1.2 Partition Level Errors

A partition level error impacts only one partition. Examples of partition level errors are:

- Partition configuration **table** error during partition initialization
- Partition initialization error
- Errors that occur during process management
- Errors that occur during error handler process

The HM will not violate partitioning when handling partition level errors

2.4.1.3 Module Level Errors

A module level error may impact all the partitions within a module. Examples of module level errors are:

- Module configuration **table** error during module initialization
- Other errors during **core** module initialization **(e.g., checksum error detected for one partition)**
- Errors during system-specific function execution
- Errors during partition switching
- Power fail

2.4.2 Fault Detection and Response

Faults are detected by several elements:

- Hardware - memory protection violation, privilege execution violation, overflow, zero divide, timer interrupt, I/O error
- **O/S** - configuration, deadline

- Application - failure of sensor, discrepancy in a multiple redundant output

The specific list of errors and where they are detected is implementation specific.

A fault response depends on the detected error and on the operational state in which the error is detected. The operational state of the system (**core** module initialization, system-specific function, partition switching, partition initialization, process management, process execution, etc.) is managed by the O/S. The response to errors in each operational state is implementation dependent. For example, a hardware failure affecting a core module I/O could either cause the O/S to shut down all the partitions immediately or signal the problem to the partition only when a process of this partition would use the failing I/O device. The system integrator chooses the error management policy.

2.4.2.1 Process Level Error Response Mechanisms

Fault responses to process level errors are determined by the application programmer using a special (highest priority**, no process identifier**) process of the partition: the error handler process. The error handler process is active in NORMAL mode only. The **error handler process** can identify the error and the faulty process via **the GET_ERROR_STATUS** service and then takes the recovery action at the process level (e.g., stop, start process) or at the partition level (e.g., set partition mode to IDLE, COLD_START, **or** WARM_START). An error code is assigned to several process level errors (e.g., numeric error corresponds to overflow, divide by zero, floating-point underflow, etc.). There is a HM service which returns the error code to the faulty application. To maintain application portability, error codes are implementation independent. **The process level** error codes and examples of each are listed below:

- Deadline_Missed - process deadline violation
- Application_Error - error raised by an application process
- Numeric_Error - during process execution, error types of overflow, divide by zero, floating-point error
- Illegal_Request - illegal O/S request by a process
- Stack_Overflow - process stack overflow
- Memory_Violation - during process execution, error types of memory protection, supervisor privilege violation
- Hardware_Fault - during process execution, error types of memory parity, I/O access error
- Power_Fail - notification of power interruption (e.g., to save application specific state data)

If an error handler process does not exist within a partition, then the partition level error response mechanism is invoked by the O/S.

Process level errors that occur inside the error handler process are always promoted to partition level (irrespective of the HM configuration) and consequently handled by the partition level error response mechanisms.

COMMENTARY

> It is possible for an application within a partition to fail in such a way that the failure is not correctly reported, or not reported at all, by the application itself (e.g., the error handler process). The system integrator may need to account for this in the overall system design (rate monitors, watchdog timers, etc.).

2.4.2.2 Partition Level Error Response Mechanisms

Fault responses to partition level errors are handled in the following way:

- **The O/S looks up the error code response action in the HM configuration tables.**
- The O/S performs the response identified in the configuration table.

2.4.2.3 Module Level Error Response Mechanisms

Fault responses to module level errors are handled in the following way:

- **The O/S looks up the error code response action in the HM configuration tables.**
- The O/S performs the response identified in the configuration table.

2.4.3 Recovery Actions

Recovery actions include actions at the **module level, partition level and process level.**

Recovery actions at all levels include the capability to ignore an error. However, an error should only be ignored if it does not leave module, partition, or process in an undefined state. For instance, ignoring memory violations may leave the partition in an undefined state.

2.4.3.1 Module Level Error Recovery Actions

The recovery action is specified for a module error in the Module HM configuration table depending on system state. Possible recovery actions are listed below:

- **Ignore**
- **Shutdown the module**
- **Reset the module**
- **Recovery Actions defined by the implementation**

The O/S must support the capability of adding platform specific recovery actions.

2.4.3.2 Partition Level Error Recovery Actions

The recovery action is specified for partition errors in the HM configuration tables. For each partition, the fault response for the error type takes into account the partition behavior capability (re-settable or not, degraded mode, etc). The recovery actions are listed below:

- **Ignore**
- Stop the partition **(IDLE)**
- Restart the partition **(COLD_START or WARM_START)**

2.4.3.3 Process Level Error Recovery Actions

The application supplier defines an error handler process of the partition for process level errors. Possible recovery actions are listed below:

- **Ignore, log the failure but take no action.**
- **Ignore the error n times before action recovery.**
- **Stop faulty process and re-initialize it from entry address.**
- **Stop faulty process and start another process.**

- **Stop faulty process (assume partition detects and recovers).**
- **Restart the partition (COLD_START or WARM_START).**
- **Stop the partition (IDLE).**

COMMENTARY

The HM design may allow support for Ada exceptions, but there is potential for conflicts between the Ada runtime system and the HM.

2.5 Configuration Considerations

It is a goal of this specification to define an environment that allows full portability and reuse of application source code. Applications must however be integrated into a system configuration which satisfies the functional requirements of the applications and meets the availability and integrity requirements of the aircraft functions. A single authoritative organization will be responsible for the integration. This organization will be known as the system integrator. The system integrator could be the **platform** supplier, the airframe manufacturer, a third party, or a combination of all participants.

The system integrator is responsible for allocating partitions to core modules. The resulting configuration should allow each partition access to its required resources and should ensure that each application's availability and integrity requirements are satisfied. The system integrator should, therefore, know the timing requirements, memory usage requirements, and external interface requirements of each partition to be integrated.

It is the responsibility of the application developer to configure the processes within a partition.

The system integrator should ensure that all partitions have access to the external data required for their correct execution. This data is passed around the system in the form of messages. Each message should, therefore, be unique and identifiable. From an application point of view, the only means of identifying a message is the port identifier, which is local to the partition. **It is the responsibility of the system integrator to ensure that message formats are compatible between the sending and receiving partitions.**

2.5.1 Configuration Information Required by the Integrator

To enable configuration of partitions onto core modules, the integrator requires the following information about each partition as a minimum:

- Memory requirements
- Period
- Duration
- **Port attributes**

Use of this information will allow the partitions to be allocated to the core modules in a configuration which ensures that the memory and time requirements of each partition can be satisfied. The location of partitions on modules dictates the routes of messages, and so the **system integrator** must also provide the mapping between nodes on the message paths.

2.5.2 Configuration Tables

Configuration tables are required by the O/S for ensuring that the system is complete during initialization and to enable communications between partitions. Configuration tables are static data areas accessed by the O/S. They cannot be accessed directly by applications, and they are not part of the O/S.

XML is used to describe the configuration as discussed in Section 5 of this document. Appendices G and H define the XML-schema types of the data needed to specify any ARINC 653 configuration. The XML-schema types must be used by the ARINC 653 O/S implementers to create the schema for their implementation.

2.5.2.1 Configuration Tables for System Initialization

(This material deleted by Supplement 3.)

2.5.2.2 Configuration Tables for Inter-Partition Communication

(This material deleted by Supplement 3.)

2.5.2.3 Configuration Tables for Health Monitor

The Health Monitor (HM) uses configuration tables to handle each occurring error. **These tables are the Module HM table, Multi-Partition HM tables and Partition HM tables.**

For the Module HM table:

- **The O/S uses the Module HM table when an error is detected outside a partition time window (e.g. during core module initialization, partition switch, etc.).**
- **The Module HM table defines the recovery action (e.g. shut down the module, reset the module, etc.) associated to module level errors detected outside a partition time window.**
- **The recovery action is set according to the detected error and the system state (e.g. init, switching partition, etc.).**
- **There is one Module HM table per module, and it is configured by the module supplier.**

For the Multi-Partition HM tables:

- **The O/S uses the Multi-Partition HM table when an error is detected inside a partition time window.**
- **The Multi-Partition HM table defines global behavior of HM for a set of partitions.**
- **The Multi-Partition HM table defines the error level (Module or Partition) associated to errors detected inside a partition time window.**
- **When the error is at module level, the Multi-Partition HM table defines also the module recovery action (e.g. shut down the module, reset the module, etc.).**
- **The error level and module recovery action (when applicable) are set according to the detected error. Since the Multi-Partition HM table is used only when one partition is inside its time window, there is only one system state (e.g. partition execution).**
- **There are one or many Multi-Partition HM tables per module, all of them configured by the system integrator. Each Multi-Partition HM table is associated to one or many partitions. These associations are configured also by the system integrator.**

For the Partition HM tables:

- **The O/S uses the Partition HM table when an error is detected inside a partition time window and the corresponding Multi-Partition HM table indicates the Partition level for this particular error in the current partition.**

- **The Partition HM table defines local behavior of HM for a single partition.**
- **The Partition HM table defines the error level (Partition or Process) associated to errors detected inside a partition time window.**
- **The Partition HM table defines also the partition recovery actions (e.g., stop the partition, restart the partition in warm or cold mode). This partition recovery action will be performed when the error is at partition level, and also when the error is at process level, but no Error Handler is present in that partition.**
- **When the error is at process level, the Partition HM table defines also the associated error codes (e.g., APPLICATION_ERROR, NUMERIC_ERROR, etc.).**
- **The error level, partition recovery action, and error codes are set according to the detected error. The list of error identifiers in the Partition HM table is restricted to the errors which can be raised by the partition itself (e.g., deadline missed, MMU violation, etc.).**
- **There is one Partition HM table per partition. Each of them is configured by the respective application supplier.**

2.6 Verification

The system integrator will be responsible for verifying that the complete system fulfills its functional requirements when applications are integrated and for ensuring that availability and integrity requirements are met. Verification that application software fulfills its functional requirements will be carried out by the supplier of the application.

3.1 Service Request Categories

This section specifies the service requests corresponding to the functions described in Section 2 of this document. The requests are grouped into the following major categories:

1. Partition Management
2. Process Management
3. Time Management
4. Memory Management
5. Interpartition Communication
6. Intrapartition Communication
7. Health Monitoring

Each service request has a sample specification written in the APEX specification grammar. The service requests in this section describe the functional requirements of APEX services. The data type names, service request names, parameter names, and order of parameters are definitive, the implementation is not **(i.e., the order of the error tests are not defined by this standard)**. Some data type declarations are common to all categories, and are duplicated in each of the following sections for clarity. The exceptions are RETURN_CODE_TYPE and SYSTEM_TIME_TYPE, which are referenced from the other sections of this document.

In order to preserve the integrity of information managed by the services, the requesting process is assumed to never be preempted during the execution of a service, except at the scheduling points which are explicitly mentioned in the semantic description.

Partition level objects **required for create services** may be statically or dynamically allocated. For a static system, the object attributes of the create service need to be checked against the corresponding statically configured objects. For a dynamic system, the object attributes need to be checked for reasonableness (e.g., range checking) and non-violation of system limits (e.g., sufficient memory available) before the object is created.

Since this interface may be applied to Integrated Modular Avionics (IMA), it does not include services (or service requirements) which only pertain to a specific architecture design, implementation, or partition configuration. This list of services is viewed as a minimum set of the total services which may be provided by the individual systems.

The interface defined in this standard provides the operations necessary for basic **multi-process execution**. The inclusion of extra services pertinent to a specific system would not necessarily violate this standard, but would make the application software which uses these additional services less portable.

The convention followed for error cases in these service requests is to not specify assignments for output parameters other than the return code. Implementers should be aware that, in many cases, programming practices will require some value to be assigned to all output parameters.

Ada and C language specifications for the types, records, constants, and service request interfaces used in this section are defined in Appendices D and E. Unless specifically annotated as implementation dependent, the values specified in these appendices should be implemented as defined. For C, the APEX specifications will be available via the “ARINC653.h” header file.

The definition of ARINC 653 NAME_TYPE is based on an ‘n-character array (i.e. fixed length).

COMMENTARY

To avoid unnecessary complexity in the underlying O/S, there are no restrictions on the allowable characters for NAME_TYPE. However, it is recommended that usage be limited to printable characters. The O/S should treat the contents of NAME_TYPE objects as case insensitive.

The use of a NULL character is not required in an object of NAME_TYPE. If a NAME_TYPE object contains NULL characters, the first NULL character in the array of characters should be considered to define the end of the name.

3.1.1 Return Code Data Type

This section contains the return code data type declaration common to all APEX services. The **return codes** are listed only in this section for clarity. The allowable return codes and their descriptions are:

```
NO_ERROR           request valid and operation performed
NO_ACTION          system's operational status unaffected by request
NOT_AVAILABLE      the request cannot be performed immediately
INVALID_PARAM      parameter specified in request invalid
INVALID_CONFIG     parameter specified in request incompatible with current
                   configuration (e.g., as specified by system integrator)
INVALID_MODE       request incompatible with current mode of operation
TIMED_OUT          time-out associated with request has expired
```

The distinction between INVALID_PARAM and INVALID_CONFIG is that INVALID_PARAM denotes invalidity regardless of the system's configuration whereas INVALID_CONFIG denotes conflict with the current integrator-specified configuration and so may change according to the degree of configuration imposed. Note that the setting of INVALID_PARAM may sometimes be redundant for strongly typed languages.

The return code data type declaration is:

```
type RETURN_CODE_TYPE      is (NO_ERROR,
                               NO_ACTION,
                               NOT_AVAILABLE,
                               INVALID_PARAM,
                               INVALID_CONFIG,
                               INVALID_MODE,
                               TIMED_OUT);
```

3.1.2 OUT Parameters Values

The validity of OUT parameters is dependent on the RETURN_CODE value returned by the considered service.

3.2 Partition Management

This section contains types and specifications related to partition management.

3.2.1 Partition Management Types

```
type OPERATING_MODE_TYPE   is (IDLE, COLD_START, WARM_START, NORMAL);
type PARTITION_ID_TYPE     is a numeric type;
type LOCK_LEVEL_TYPE       is a numeric type;

type START_CONDITION_TYPE  is (NORMAL_START,
                               PARTITION_RESTART,
                               HM_MODULE_RESTART,
                               HM_PARTITION_RESTART);
```

Where:

```
NORMAL_START          is a normal power-up.
PARTITION_RESTART     is either due to COLD_START or WARM_START by the partition
                      itself, through the SET_PARTITION_MODE service.
HM_MODULE_RESTART     is a recovery action taken at module level by the HM.
HM_PARTITION_RESTART  is a recovery action taken at partition level by the HM.
```

The NORMAL_START/PARTITION_RESTART can be set if there is a manual module/partition reset through external means.

```
type PARTITION_STATUS_TYPE is record
   IDENTIFIER           : PARTITION_ID_TYPE;
   PERIOD               : SYSTEM_TIME_TYPE;
   DURATION             : SYSTEM_TIME_TYPE;
   LOCK_LEVEL           : LOCK_LEVEL_TYPE;
   OPERATING_MODE       : OPERATING_MODE_TYPE;
   START_CONDITION      : START_CONDITION_TYPE;
end record;
```

The RETURN_CODE_TYPE is common to all APEX services and is defined in Section 3.1.1 of this document.

The SYSTEM_TIME_TYPE is common to many APEX services and is defined in Section 3.4.1 of this document.

3.2.2 Partition Management Services

The partition management services are:

```
   GET_PARTITION_STATUS
   SET_PARTITION_MODE
```

3.2.2.1 GET_PARTITION_STATUS

The GET_PARTITION_STATUS service request is used to obtain the status of the current partition.

```
procedure GET_PARTITION_STATUS
   (PARTITION_STATUS : out PARTITION_STATUS_TYPE;
    RETURN_CODE      : out RETURN_CODE_TYPE) is

normal
   PARTITION_STATUS  := current value of partition status;
   RETURN_CODE       := NO_ERROR;

end GET_PARTITION_STATUS;
```

The return codes for this service are explained below.

GET_PARTITION_STATUS

Return Code Value	Commentary
NO_ERROR	Successful completion

3.2.2.2 SET_PARTITION_MODE

The SET_PARTITION_MODE service request is used to set the operating mode of the current partition to normal after the application portion of the initialization of the partition is complete. The service is also used for setting the partition back to idle (partition shutdown), and to cold start or warm start (partition restart), when a fault is detected and processed.

```
procedure SET_PARTITION_MODE
   (OPERATING_MODE : in  OPERATING_MODE_TYPE;
    RETURN_CODE    : out RETURN_CODE_TYPE) is

error
   when (OPERATING_MODE does not represent an existing mode) =>
      RETURN_CODE := INVALID_PARAM;
   when (OPERATING_MODE is NORMAL and current mode is NORMAL) =>
      RETURN_CODE := NO_ACTION;
   when (OPERATING_MODE is WARM_START and current mode is COLD_START) =>
      RETURN_CODE := INVALID_MODE;

normal
   set current partition’s operating mode := OPERATING_MODE;
   if (OPERATING_MODE is IDLE) then
      shut down the partition;
   end if;
   if (OPERATING_MODE is WARM_START or OPERATING_MODE is COLD_START) then
      inhibit process scheduling and switch back to initialization mode;
   end if;
   if (OPERATING_MODE is NORMAL) then
      set to READY all previously started (not delayed) aperiodic processes
      (unless the process was suspended);
      set release point of all previously delay started aperiodic processes
      to the system clock time plus their delay times;
      set first release points of all previously started (not delayed) periodic
      processes to the partition’s next periodic processing start;
      set first release points of all previously delay started periodic processes
      to the partition’s next periodic processing start plus their delay times;
      -- at their release points, the processes are set to READY (if not DORMANT)
      calculate the DEADLINE_TIME of all non-dormant processes in the partition;
      -- a DEADLINE_TIME calculation may cause an overflow of the underlying
      -- clock. If this occurs, HM is invoked with an illegal request error code
      set the partition’s lock level to zero;
      if (an error process has been created) then
         enable the error process for execution and fault processing;
      end if;
      activate the process scheduling;
   end if;
   RETURN_CODE := NO_ERROR;

end SET_PARTITION_MODE;
```

The return codes for this service are explained below.

SET_PARTITION_MODE

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	OPERATING_MODE does not represent an existing mode
NO_ACTION	OPERATING_MODE is normal and current mode is NORMAL
INVALID_MODE	OPERATING_MODE is WARM_START and current mode is COLD_START

COMMENTARY

It is O/S implementation dependent whether the process used during the **partition** initialization phase continues to run after setting OPERATING_MODE to NORMAL. From the viewpoint of portability, the application should not depend on the process continuing to run.

3.3 Process Management

This section contains types and specifications related to process management. **The scope of a process management service is restricted to a partition.**

3.3.1 Process Management Types

```
type PROCESS_ID_TYPE     is a numeric type;
type PROCESS_NAME_TYPE   is a n-character string;
type PRIORITY_TYPE       is a numeric type;
type STACK_SIZE_TYPE     is a numeric type;
type LOCK_LEVEL_TYPE     is a numeric type;
type SYSTEM_ADDRESS_TYPE is language dependent;
type PROCESS_STATE_TYPE  is (DORMANT, READY, RUNNING, WAITING);
type DEADLINE_TYPE       is (SOFT, HARD);

type PROCESS_ATTRIBUTE_TYPE is record
   NAME             : PROCESS_NAME_TYPE;
   ENTRY_POINT      : SYSTEM_ADDRESS_TYPE;
   STACK_SIZE       : STACK_SIZE_TYPE;
   BASE_PRIORITY    : PRIORITY_TYPE;
   PERIOD           : SYSTEM_TIME_TYPE;
   TIME_CAPACITY    : SYSTEM_TIME_TYPE;
   DEADLINE         : DEADLINE_TYPE;
end record;

type PROCESS_STATUS_TYPE is record
   ATTRIBUTES       : PROCESS_ATTRIBUTE_TYPE;
   CURRENT_PRIORITY : PRIORITY_TYPE;
   DEADLINE_TIME    : SYSTEM_TIME_TYPE;
   PROCESS_STATE    : PROCESS_STATE_TYPE;
end record;
```

The RETURN_CODE_TYPE is common to all APEX services and is defined in Section 3.1.1 of this document.

The SYSTEM_TIME_TYPE is common to many APEX services and is defined in Section 3.4.1 of this document.

3.3.2 Process Management Services

A process must be created during the **partition** initialization phase before it can be used. The process management services are:

```
 GET_PROCESS_ID
 GET_PROCESS_STATUS
 CREATE_PROCESS
 SET_PRIORITY
 SUSPEND_SELF
 SUSPEND
 RESUME
 STOP_SELF
 STOP
 START
 DELAYED_START
 LOCK_PREEMPTION
 UNLOCK_PREEMPTION
 GET_MY_ID
```

3.3.2.1 GET_PROCESS_ID

The GET_PROCESS_ID service request allows a process to obtain a process identifier by specifying the process name.

```
procedure GET_PROCESS_ID
   (PROCESS_NAME  : in  PROCESS_NAME_TYPE;
    PROCESS_ID    : out PROCESS_ID_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (there is no current partition process named PROCESS_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

normal
   PROCESS_ID  := (ID of the process named PROCESS_NAME);
   RETURN_CODE := NO_ERROR;

end GET_PROCESS_ID;
```

The return codes for this service are explained below.

GET_PROCESS_ID

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current partition process named PROCESS_NAME

3.3.2.2 GET_PROCESS_STATUS

The GET_PROCESS_STATUS service request returns the current status of the specified process. The current status of each of the individual processes of a partition is available to all processes within that partition.

```
procedure GET_PROCESS_STATUS
   (PROCESS_ID     : in  PROCESS_ID_TYPE;
    PROCESS_STATUS : out PROCESS_STATUS_TYPE;
    RETURN_CODE    : out RETURN_CODE_TYPE) is

error
   when (PROCESS_ID does not identify an existing process) =>
      RETURN_CODE := INVALID_PARAM;

normal
   RETURN_CODE    := NO_ERROR;
   PROCESS_STATUS := current value of process status;

end GET_PROCESS_STATUS;
```

The return codes for this service are explained below.

GET_PROCESS_STATUS

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	PROCESS_ID does not identify an existing process

3.3.2.3 CREATE_PROCESS

The CREATE_PROCESS service request creates a process and returns an identifier that denotes the created process. **For each partition, as many processes as the pre-allocated memory space will support can be created. The creation of processes (e.g., names used, number of processes) for one partition has no impact on the creation of processes for other partitions.**

Defining the PERIOD of a process with an INFINITE_TIME_VALUE inherently defines an aperiodic process.

Defining the TIME_CAPACITY of a process with an INFINITE_TIME_VALUE inherently defines a process **that does not have a deadline time (i.e., will not cause a DEADLINE_MISSED HM event). Deadline times can be defined for periodic and aperiodic processes.**

```
procedure CREATE_PROCESS
   (ATTRIBUTES  : in  PROCESS_ATTRIBUTE_TYPE;
    PROCESS_ID  : out PROCESS_ID_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (insufficient storage capacity for the creation of the specified
      process or maximum number of processes have been created) =>
      RETURN_CODE := INVALID_CONFIG;
   when (the process named ATTRIBUTES.NAME is already created) =>
      RETURN_CODE := NO_ACTION;
   when (ATTRIBUTES.STACK_SIZE out of range) =>
      RETURN_CODE := INVALID_PARAM;
   when (ATTRIBUTES.BASE_PRIORITY out of range) =>
      RETURN_CODE := INVALID_PARAM;
   when (ATTRIBUTES.PERIOD out of range) =>
      RETURN_CODE := INVALID_PARAM;
   when (ATTRIBUTES.PERIOD is finite (i.e., periodic process) and
      ATTRIBUTES.PERIOD is not an integer multiple of partition period defined
      in the configuration tables) =>
      RETURN_CODE := INVALID_CONFIG;
   when (ATTRIBUTES.TIME_CAPACITY out of range) =>
      RETURN_CODE := INVALID_PARAM;
   when (ATTRIBUTES.PERIOD is finite (i.e., periodic process) and
      ATTRIBUTES.TIME_CAPACITY is greater than ATTRIBUTES.PERIOD) =>
      RETURN_CODE := INVALID_PARAM;
   when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

normal
   create a new process with the process attributes set to ATTRIBUTES;
   set the process state to DORMANT;
   initialize process context and stack;
   PROCESS_ID  := unique identifier assigned by the O/S to the created process;
   RETURN_CODE := NO_ERROR;

end CREATE_PROCESS;
```

The return codes for this service are explained below.

CREATE_PROCESS

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	**Insufficient storage capacity for the creation of the specified process or maximum number of processes have been created**
NO_ACTION	The process named ATTRIBUTES.NAME is already created
INVALID_PARAM	ATTRIBUTES.STACK_SIZE out of range
INVALID_PARAM	ATTRIBUTES.BASE_PRIORITY out of range
INVALID_PARAM	ATTRIBUTES.PERIOD **out of range**
INVALID_CONFIG	**ATTRIBUTES.PERIOD is finite and ATTRIBUTES.PERIOD is not an integer multiple of partition period defined in the configuration tables**
INVALID_PARAM	ATTRIBUTES.TIME_CAPACITY out of range

INVALID_PARAM	ATTRIBUTES.PERIOD is finite and ATTRIBUTES.TIME_CAPACITY is greater than ATTRIBUTES.PERIOD
INVALID_MODE	Operating mode is NORMAL

3.3.2.4 SET_PRIORITY

The SET_PRIORITY service request changes a process's current priority. **If the process's current state is ready, the process is considered as having been in the ready state with the set priority for the shortest elapsed time (i.e., all other processes at the same priority will be selected to run before this process).**

Process rescheduling is performed after this service request only when the process whose priority is changed is in the ready or running state.

COMMENTARY

> If the process is waiting on a resource, the process queue for that resource may or may not be reordered based on the new priority. Therefore, from the viewpoint of portability, applications should not rely on the reordering of the process queue.

```
procedure SET_PRIORITY
   (PROCESS_ID     : in  PROCESS_ID_TYPE;
    PRIORITY       : in  PRIORITY_TYPE;
    RETURN_CODE    : out RETURN_CODE_TYPE) is

error
   when (PROCESS_ID does not identify an existing process) =>
      RETURN_CODE := INVALID_PARAM;
   when (PRIORITY is out of range) =>
      RETURN_CODE := INVALID_PARAM;
   when (specified process is in the DORMANT state) =>
      RETURN_CODE := INVALID_MODE;

normal
   set the current priority of the specified process to PRIORITY;
   if (preemption is enabled) then
      ask for process scheduling;
      -- The current process may be preempted by the process whose priority
      -- was modified
   end if;
   RETURN_CODE := NO_ERROR;

end SET_PRIORITY;
```

The return codes for this service are explained below.

SET_PRIORITY

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	PROCESS_ID does not identify an existing process
INVALID_PARAM	PRIORITY is out of range
INVALID_MODE	The specified process is in the DORMANT state

3.3.2.5 SUSPEND_SELF

The SUSPEND_SELF service request suspends the execution of the current process, if aperiodic. **The process remains suspended** until the RESUME service request is issued or the specified time-out value expires.

Periodic processes cannot be suspended.

COMMENTARY

> If a process suspends (using SUSPEND service) an already self-suspended process, it has no effect.

```
procedure SUSPEND_SELF
   (TIME_OUT     : in  SYSTEM_TIME_TYPE;
    RETURN_CODE  : out RETURN_CODE_TYPE) is

error
   when (preemption is disabled or process is error handler process) =>
      RETURN_CODE := INVALID_MODE;
   when (TIME_OUT is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;
   when (process is periodic) =>
      RETURN_CODE := INVALID_MODE;

normal
   if (TIME_OUT is zero) then
      RETURN_CODE := NO_ERROR;
   else
      set the current process's state to WAITING;
      if (TIME_OUT is not infinite) then
         initiate a time counter with duration TIME_OUT;
      end if;
      ask for process scheduling;
      -- The current process is blocked and will return in READY state
      -- by expiration of time-out or by RESUME service request from another
      -- process
      if (expiration of the time-out) then
         RETURN_CODE := TIMED_OUT;
      else  -- RESUME service request from another process
         RETURN_CODE := NO_ERROR;
         -- RESUME service request will stop the time counter.
      end if;
   end if;

end SUSPEND_SELF;
```

The return codes for this service are explained below.

SUSPEND_SELF

Return Code Value	Commentary
NO_ERROR	**Resumed by another process or TIME_OUT is zero**
INVALID_MODE	Preemption is disabled or process is error handler process
INVALID_PARAM	**TIME_OUT is out of range**
INVALID_MODE	Process is periodic
TIMED_OUT	**The specified time-out is expired**

3.3.2.6 SUSPEND

The SUSPEND service request allows the current process to suspend the execution of any aperiodic process except itself. The suspended process **remains suspended until** resumed by another process. If the process is pending in a queue at the time it is suspended, it is not removed from that queue. When it is resumed, it will continue pending unless it has been removed from the queue (by **availability of the resource it was waiting on** or expiration of a time-out) before the end of its suspension. **If the process is waiting on a timed-wait time counter, the process will continue to wait on the time counter when it is resumed unless the time counter has expired.**

Periodic processes cannot be suspended.

COMMENTARY

> If a process suspends an already suspended process, it has no effect.
>
> **A process may suspend any other process asynchronously. Though this practice is not recommended, there may be partitions that require this capability.**

```
procedure SUSPEND
  (PROCESS_ID   : in  PROCESS_ID_TYPE;
   RETURN_CODE : out RETURN_CODE_TYPE) is

error
   -- The error handler process cannot suspend the process which has
   -- called LOCK_PREEMPTION in the case where the error handler was
   -- invoked while lock level is greater than zero (i.e., preemption disabled).
   when (preemption is disabled and the PROCESS_ID is the process which
      the error handler has pre-empted) =>
      RETURN_CODE := INVALID_MODE;
   when (PROCESS_ID does not identify an existing process or identifies the current
      process) =>
      RETURN_CODE := INVALID_PARAM;
   when (the state of the specified process is DORMANT) =>
      RETURN_CODE := INVALID_MODE;
   when (specified process is periodic) =>
      RETURN_CODE := INVALID_MODE;

normal
   if (specified process has already been suspended,
       either through SUSPEND or SUSPEND_SELF) then
      RETURN_CODE := NO_ACTION;
   else
      set the specified process state to WAITING;
      RETURN_CODE := NO_ERROR;
   end if;

end SUSPEND;
```

The return codes for this service are explained below.

SUSPEND

Return Code Value	Commentary
NO_ERROR	Successful completion
NO_ACTION	Specified process has already been suspended
INVALID_PARAM	PROCESS_ID does not identify an existing process or identifies **the current process**
INVALID_MODE	preemption is disabled and the PROCESS_ID is the process which the error handler has pre-empted
INVALID_MODE	The state of the specified process is DORMANT
INVALID_MODE	Specified process is periodic

3.3.2.7 RESUME

The RESUME service request allows the current process to resume a previously suspended process **(i.e., via SUSPEND or SUSPEND_SELF services)**. The resumed process will become ready if it is not waiting on a **process queue (e.g., resource associated with a queuing port, buffer, semaphore, event) or a time delay associated with the TIMED_WAIT service.**

A periodic process cannot be suspended, so it cannot be resumed.

```
procedure RESUME
   (PROCESS_ID  : in  PROCESS_ID_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (PROCESS_ID does not identify an existing process or identifies the current
         process) =>
         RETURN_CODE := INVALID_PARAM;
   when (the state of the specified process is DORMANT) =>
         RETURN_CODE := INVALID_MODE;
   when (PROCESS_ID identifies a periodic process) =>
         RETURN_CODE := INVALID_MODE;
   when (identified process is not a suspended process) =>
         RETURN_CODE := NO_ACTION;

normal
   if (the specified process was suspended with a time-out) then
      stop the affected time counter;
   end if;
   if (the specified process is not waiting on a process queue or TIMED_WAIT
      time delay) then
      set the specified process state to READY;
      if (preemption is enabled) then
         ask for process scheduling;
         -- The current process may be preempted by the resumed process
      end if;
   end if;
   RETURN_CODE := NO_ERROR;

end RESUME;
```

The return codes for this service are explained below.

RESUME

Return Code Value	Commentary
NO_ERROR	Successful completion
NO_ACTION	Identified process is not a suspended process
INVALID_PARAM	PROCESS_ID does not identify an existing process or identifies **the current process**
INVALID_MODE	The state of the specified process is DORMANT
INVALID_MODE	PROCESS_ID identifies a periodic process

3.3.2.8 STOP_SELF

The STOP_SELF service request allows the current process to stop itself. If the current process is not the error handler process, **the partition’s lock level is reset (i.e., preemption is enabled).**

No return code is returned to the requesting process procedure.

COMMENTARY

This service should not be called when the partition is in the WARM_START or the COLD_START mode. In this case, the behavior of this service is not defined.

```
procedure STOP_SELF is

normal
   if (not error handler process) then
      reset the partition’s LOCK_LEVEL counter (i.e., enable preemption);
   end if;
   set the current process state to DORMANT;
   if (current process is the error handler process and preemption is disabled
   and previous process is not stopped) then
      return to previous process;
   else
      ask for process scheduling;
   end if;

end STOP_SELF;
```

3.3.2.9 STOP

The STOP service request makes a process ineligible for processor resources until another process issues the START service request.

This service allows the current process to **stop** the execution of any process except itself. When a process **stops** another process that is **currently waiting in a process queue, the stopped process is removed from the process queue.**

```
procedure STOP
   (PROCESS_ID   : in  PROCESS_ID_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (PROCESS_ID does not identify an existing process or identifies the current
      process) =>
      RETURN_CODE := INVALID_PARAM;
   when (the state of the specified process is DORMANT) =>
      RETURN_CODE := NO_ACTION;

normal
   set the specified process state to DORMANT;
   if (current process is error handler and PROCESS_ID is process which the error
      handler preempted) then
      reset the partition’s LOCK_LEVEL counter (i.e., enable preemption);
   end if;
   if (specified process is waiting in a process queue) then
      remove the process from the process queue;
   end if;
   stop any time counters associated with the specified process;
   RETURN_CODE := NO_ERROR;

end STOP;
```

The return codes for this service are explained below.

STOP

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	PROCESS_ID does not identify an existing process or identifies **the current process**
NO_ACTION	The state of the specified process is DORMANT

COMMENTARY

> Stopping a process that has locked preemption (e.g., by the error handler) may result in improper operation of the application.

3.3.2.10 START

The START service request initializes all attributes of a process to their default values and resets the runtime stack of the process. If the partition is in the NORMAL mode, the process' deadline expiration time and next release point are calculated.

This service allows the current process to start the execution of another process during runtime.

```
procedure START
   (PROCESS_ID   : in  PROCESS_ID_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (PROCESS_ID does not identify an existing process) =>
      RETURN_CODE := INVALID_PARAM;
   when (the state of the specified process is not DORMANT) =>
      RETURN_CODE := NO_ACTION;
   when (DEADLINE_TIME calculation is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_CONFIG;

normal
   if (the process is an aperiodic process) then
      -- Start_aperiodic_process
      set the current priority of specified process to its base priority;
      reset context and stack;
      if (operating mode is NORMAL) then
         set the specified process state to READY;
         set the DEADLINE_TIME value for the specified process to
         (current system clock plus TIME_CAPACITY);
         if (preemption is enabled) then
            ask for process scheduling;
            -- The calling process may be preempted
         end if;
      else
         -- (the mode of the partition is COLD_START or WARM_START)
         set the specified process state to WAITING;
         -- The specified process will wait until NORMAL mode
      end if;
      RETURN_CODE := NO_ERROR;
      -- At the end of the initialization phase (when the partition mode
      -- becomes NORMAL) the process is set to READY in accordance with
     --  description made in SET_PARTITION_MODE. Process states are set in
     -- accordance with Section 2.3.2.2.1.3, State Transitions

      -- End of Start_aperiodic_process
   else
      -- Start_periodic_process
      set the current priority of specified process to its base priority;
      reset context and stack;
      if (operating mode is NORMAL) then
         set the specified process state to WAITING;
         set the first release point of the specified process;
         set the DEADLINE_TIME value for the specified process to
         (first release point plus TIME_CAPACITY);
         -- Specified process has to wait for its release point
      else
         -- (the mode of the partition is COLD_START or WARM_START)
         set the specified process state to WAITING;
         -- The specified process will wait until NORMAL mode
      end if;
      RETURN_CODE := NO_ERROR;
      -- At the end of the initialization phase (when the partition mode
      -- becomes NORMAL) the first release point and first DEADLINE_TIME
      -- are computed in accordance with description made in SET_PARTITION_MODE.
      -- Process states are set in accordance with Section 2.3.2.2.1.3, State
      -- Transitions
      -- End of Start_periodic_process
   end if;

end START;
```

The return codes for this service are explained below.

START

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	PROCESS_ID does not identify an existing process
INVALID_CONFIG	**DEADLINE_TIME calculation is out of range**
NO_ACTION	The state of the specified process is not DORMANT

3.3.2.11 DELAYED_START

The DELAYED_START service request initializes all attributes of a process to their default values, resets the runtime stack of the process, and places the process into the waiting state, i.e., the specified process goes from dormant to waiting. If the partition is in the normal operating mode, the process's release point is calculated with the specified delay time, and the process's deadline expiration time is also calculated.

This service allows the current process to start the execution of another process during runtime.

COMMENTARY

DELAYED_START with a delay time of zero is equivalent to START.

```
procedure DELAYED_START
   (PROCESS_ID  : in  PROCESS_ID_TYPE;
    DELAY_TIME  : in  SYSTEM_TIME_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (PROCESS_ID does not identify an existing process) =>
      RETURN_CODE := INVALID_PARAM;
   when (the state of the specified process is not DORMANT) =>
      RETURN_CODE := NO_ACTION;
   when (DELAY_TIME is out of range) =>
      RETURN_CODE := INVALID_PARAM;
   when (DELAY_TIME is infinite) =>
      RETURN_CODE := INVALID_PARAM;   when (the process is periodic and
      DELAY_TIME is greater or equal to the period of the specified process) =>
      RETURN_CODE := INVALID_PARAM;
   when (DEADLINE_TIME calculation is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_CONFIG;

normal
   if (the specified process is an aperiodic process) then
      -- Start_aperiodic_process with delay_time
      set the current priority of specified process to its base priority;
      reset context and stack;
      if (operating mode is NORMAL) then
         if (DELAY_TIME = 0) then
            set the specified process state to READY;
            set the DEADLINE_TIME value for the specified process to
            (current system clock plus TIME_CAPACITY);
         else
            set the specified process state to WAITING;
            set the DEADLINE_TIME value for the specified process to
            (current system clock plus TIME_CAPACITY plus DELAY_TIME);
            initiate a time counter with duration DELAY_TIME;
            -- The started process is WAITING and will go to the READY
            -- state by expiration of the delay time
         end if;
         if (preemption is enabled) then
            ask for process scheduling;
            -- The calling process may be preempted by the newly started process
         end if;
      else
         -- (the mode of the partition is COLD_START or WARM_START)
         set the specified process state to WAITING;
         -- The specified process will wait until NORMAL mode
      end if;
      RETURN_CODE := NO_ERROR;
      -- At the end of the initialization phase (when the partition mode
      -- becomes NORMAL) the release point and DEADLINE_TIME
      -- are computed in accordance with description made in SET_PARTITION_MODE.
      -- Process states are set in accordance with Section 2.3.2.2.1.3, State
      -- Transitions
      -- End of Start_aperiodic_process with delay_time
   else
      -- Start_periodic_process with delay_time
      set the current priority of specified process to its base priority;
      reset context and stack;
      if (operating mode is NORMAL) then
         set the specified process state to WAITING;
         set the first release point (see Section 2.3.1) of the specified
         process including the delay time;
         set the DEADLINE_TIME value for the specified process to
         (first release point + TIME_CAPACITY);
         -- Specified process has to wait for its release point
      else
         -- (the mode of the partition is COLD_START or WARM_START)
         set the specified process state to WAITING;
         -- The specified process will be in a waiting state until
         -- NORMAL mode is selected
      end if;
      RETURN_CODE := NO_ERROR;
      -- At the end of the initialization phase (when the partition mode
      -- becomes NORMAL) the first release point and first DEADLINE_TIME
      -- are computed in accordance with description made in SET_PARTITION_MODE.
      -- Process states are set in accordance with Section 2.3.2.2.1.3 State
      -- Transitions
      -- End of Start_periodic_process with delay_time
   end if;

end DELAYED_START;
```

The return codes for this service are explained below.

DELAYED_START

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	PROCESS_ID does not identify an existing process
INVALID_PARAM	**DELAY_TIME is out of range**
INVALID_PARAM	**DELAY_TIME is infinite**
INVALID_PARAM	**The process is periodic and** DELAY_TIME is greater or equal to the period of the specified process
INVALID_CONFIG	**DEADLINE_TIME calculation is out of range**
NO_ACTION	The state of the specified process is not DORMANT

3.3.2.12 LOCK_ PREEMPTION

The LOCK_PREEMPTION service request increments the lock level of a partition and disables process rescheduling **(i.e., preemption is disabled)** for a partition.

COMMENTARY

> This service has no impact on the scheduling of other partitions. It only affects scheduling of processes within a partition.
>
> Preemptibility control allows critical sections of code to be guaranteed to execute without preemption by other processes within the partition. **Unless a partition is restarted, if a process invoked LOCK_PREEMPTION and is interrupted by the end of a partition window, it is guaranteed to be the first to execute in the next partition window.**

```
procedure LOCK_PREEMPTION
   (LOCK_LEVEL    : out LOCK_LEVEL_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (current process is error handler process or OPERATING_MODE is not
      NORMAL) =>
      RETURN_CODE := NO ACTION;
   when (partition's lock level is greater or equal to MAX_LOCK_LEVEL) =>
      RETURN_CODE := INVALID_CONFIG;

normal
   increment the partition's lock level value;
   LOCK_LEVEL = current value of the partition's lock level;
   RETURN_CODE := NO_ERROR;

end LOCK_PREEMPTION;
```

The return codes for this service are explained below.

LOCK_PREEMPTION

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	**Partition’s lock level value is greater or equal to** MAX_LOCK_LEVEL
NO ACTION	Current process is the error handler process or OPERATING_MODE is not NORMAL

3.3.2.13 UNLOCK_PREEMPTION

The UNLOCK_PREEMPTION service request decrements the current lock level of a partition. Process scheduling is **enabled (i.e., preemption is enabled) only when the partition’s** lock level becomes zero.

```
procedure UNLOCK_PREEMPTION
   (LOCK_LEVEL   : out LOCK_LEVEL_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (the current process is the error handler process or OPERATING_MODE is
      not NORMAL or the partition’s lock level indicates unlocked (i.e., is
      zero)) =>
      RETURN_CODE := NO_ACTION;

normal
   decrement the partition’s lock level value;
   if (the partition’s lock level is zero) then
      -- Preemption is enabled
      ask for process scheduling;
      -- The current process may be preempted
   end if;
   LOCK_LEVEL = current value of the partition’s lock level;
   RETURN_CODE := NO_ERROR;

end UNLOCK_ PREEMPTION;
```

The return codes for this service are explained below.

UNLOCK_PREEMPTION

Return Code Value	Commentary
NO_ERROR	Successful completion NO_ACTION The current process is the error handler process or OPERATING_MODE is not NORMAL **or the partition’s lock level indicates unlocked (i.e., is zero)**

3.3.2.14 GET_MY_ID

The GET_MY_ID service request returns the process identifier of the current process.

```
procedure GET_MY_ID
   (PROCESS_ID    : out PROCESS_ID_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (current process has no identifier, e.g., is the error handler) =>
      RETURN_CODE := INVALID_MODE;

normal
   PROCESS_ID  := ID of the current process;
   RETURN_CODE := NO_ERROR;

end GET_MY_ID;
```

The return codes for this service are explained below.

GET_MY_ID

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_MODE	Current process has no **identifier**

3.4 Time Management

Processes that are required to execute at a particular frequency are known as periodic processes. Similarly, those processes that execute only after a particular event are known as aperiodic or event-driven processes. The Time Management services provide a means to control periodic and aperiodic processes.

For periodic processes, each process within a partition is able to specify **a maximum elapsed time for executing its processing cycle**. This time capacity is used to set a processing deadline time. The deadline time is then periodically evaluated by the operating system to determine whether the process is satisfactorily completing its processing within the allotted time. It is expected that at the end of each processing cycle a process will call the PERIODIC_WAIT service to get a new deadline. The new deadline is calculated from the next periodic release point for that process.

For all processes, the TIMED_WAIT service allows the process to suspend itself for a minimum amount of elapsed time. After the wait time has elapsed, the process becomes available to be scheduled.

The REPLENISH service allows a process to postpone its current deadline by the amount of time which has already passed.

This section contains types and specifications related to time management.

3.4.1 Time Management Types

The system time type represents duration. The value returned from the GET_TIME service will indicate the time (duration) since core module power on. When this time type is used in other time services (such as time wait) it will represent time duration, not an absolute time. Infinite time is defined as any negative value (-1 is used for the constant definition, however all negative values should be treated as INFINITE_TIME_VALUE.)

```
type SYSTEM_TIME_TYPE is a numeric type;  -- 64 bit  signed integer value with a
                                          -- 1 nanosecond LSB.
```

COMMENTARY

> The implementation of SYSTEM TIME TYPE on processors that do not natively support 64-bit signed integers may require a set of support routines to perform 64-bit signed integer operations. Software certification considerations may be necessary for these support routines. The considerations should include whether the support routines are inlined by the compiler for each instance a 64-bit operation is used or are a set of routines invoked by the compiler whenever a 64-bit signed integer operation is required.

3.4.2 Time Management Services

The time management services are:

```
TIMED_WAIT
PERIODIC_WAIT
GET_TIME
REPLENISH
```

3.4.2.1 TIMED_WAIT

The TIMED_WAIT service request suspends execution of the requesting process for a minimum amount of elapsed time. A delay time of zero allows round-robin scheduling of processes of the same priority**. When delay time is zero, all other ready processes with the current priority equal to the current process's value will be eligible to run before the current process**.

```
procedure TIMED_WAIT
   (DELAY_TIME   : in  SYSTEM_TIME_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (preemption is disabled or process is error handler process) =>
      RETURN_CODE := INVALID_MODE;
   when (DELAY_TIME is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;
   when (DELAY_TIME is infinite) =>
      RETURN_CODE := INVALID_PARAM;

normal
   if (DELAY_TIME = 0) then

      cause all other ready processes whose current priority is equal to the
      current process's value to be eligible to run before the current process;
      ask for process scheduling;
      -- Current process continues to run if there are no other processes with
      -- the same current priority or other processes with the same current
      -- priority are not in the READY state
   else
      set the current process state to WAITING;
      initiate a time counter for the current process with duration DELAY_TIME;
      ask for process scheduling;
      -- The current process is blocked and will return in READY state by
      -- expiration of the delay time (unless another process suspended it)
   end if;
   RETURN_CODE := NO_ERROR;
end TIMED_WAIT;
```

The return codes for this service are explained below.

TIMED_WAIT

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_MODE	Preemption is disabled or process is error handler process
INVALID_PARAM	DELAY_TIME is out of range
INVALID_PARAM	DELAY_TIME is infinite

3.4.2.2 PERIODIC_WAIT

The PERIODIC_WAIT service request suspends execution of the requesting **periodic** process until the next release point in the processor time line that corresponds to the period of the process.

COMMENTARY

Periodic processes should suspend their execution until periodic release points in the processor time line.

If a periodic process is waiting for its next release point and another process stops it, the activities associated with stopping the process will cause the process to no longer be waiting for the next release point.

Implementations should ensure that management of time does not result in drift between partition and process scheduling.

```
procedure PERIODIC_WAIT
   (RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (preemption is disabled or process is error handler process) =>
      RETURN_CODE := INVALID_MODE;
   when (current process is not periodic) =>
      RETURN_CODE := INVALID_MODE;
   when (DEADLINE_TIME calculation is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_CONFIG;

normal
   set the current process state to WAITING;
   Next release point := process period plus previous release point;
   -- Computation of first release point is done in SET_PARTITION_MODE service
   DEADLINE_TIME := time of next release point + TIME_CAPACITY;
   ask for process scheduling;
   -- The process is blocked until the next release point
   -- at which the process will return in READY state (unless
   -- stopped while waiting)
   RETURN_CODE := NO_ERROR;

end PERIODIC_WAIT;
```

The return codes for this service are explained below.

PERIODIC_WAIT

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_MODE	Preemption is disabled or process is error handler process
INVALID_MODE	Current process is not periodic
INVALID_CONFIG	**DEADLINE_TIME calculation is out of range**

3.4.2.3 GET_TIME

The service GET_TIME requests the current value of the system clock. The system clock is the value of a clock common to all **partitions in the core** module.

```
procedure GET_TIME
   (SYSTEM_TIME : out SYSTEM_TIME_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

normal
   SYSTEM_TIME := current system clock value;
   RETURN_CODE := NO_ERROR;

end GET_TIME;
```

The return codes for this service are explained below.

GET_TIME	
Return Code Value	Commentary
NO_ERROR	Successful completion

3.4.2.4 REPLENISH

The REPLENISH service request updates the deadline of the current process with a specified BUDGET_TIME value.

A periodic process' deadline **cannot be postponed** past its next release point.

```
procedure REPLENISH
   (BUDGET_TIME   : in SYSTEM_TIME_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (process is error handler process or OPERATING_MODE is not NORMAL) =>
      RETURN_CODE := NO_ACTION;
   when (current process is periodic and new deadline will exceed next release point) =>
      RETURN_CODE := INVALID_MODE;
   when (BUDGET_TIME is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;

normal
   set a new DEADLINE_TIME value for the current process to
   (current system clock + BUDGET_TIME);
   -- if current process's TIME_CAPACITY is infinite or BUDGET_TIME is
   -- infinite then the DEADLINE_TIME is infinite,
   -- i.e., there is no effective deadline
   RETURN_CODE := NO_ERROR;

end REPLENISH;
```

The return codes for this service are explained below.

REPLENISH	
Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	BUDGET_TIME is out of range
INVALID_MODE	**Current process is periodic and** new deadline would exceed the next release point
NO_ACTION	Process is error handler process or OPERATING_MODE is not NORMAL

3.5 Memory Management

There are no memory management services, because partitions and their associated memory spaces are defined **as part of system configuration activities. It is expected that the adequacy of the allocated memory space is confirmed as part of system build activities or at run-time before the first application process becomes running (e.g., memory resources should not be exhausted before transitioning to NORMAL).**

3.6 Interpartition Communication

This section contains types and specifications related to interpartition communication.

3.6.1 Interpartition Communication Types

These types are used in inter-partition communication services.

```
type MESSAGE_ADDR_TYPE        is a contiguous area of data defined by a starting
                              address;
type MESSAGE_SIZE_TYPE        is a numeric type; -- number of bytes
type PORT_DIRECTION_TYPE      is (SOURCE, DESTINATION);
```

The RETURN_CODE_TYPE is common to all APEX services and is defined in Section 3.1.1 of this document.

The SYSTEM_TIME_TYPE is common to many APEX services and is defined in Section 3.4.1 of this document.

These types are used in sampling port services:

```
type SAMPLING_PORT_ID_TYPE     is a numeric type;
type SAMPLING_PORT_NAME_TYPE   is a n-character string;
type VALIDITY_TYPE             is (INVALID, VALID);
type SAMPLING_PORT_STATUS_TYPE is record
   MAX_MESSAGE_SIZE   : MESSAGE_SIZE_TYPE;
   PORT_DIRECTION     : PORT_DIRECTION_TYPE;
   REFRESH_PERIOD     : SYSTEM_TIME_TYPE;
   LAST_MSG_VALIDITY  : VALIDITY_TYPE;
end record;
```

These types are used in queuing port services:

```
type QUEUING_PORT_ID_TYPE      is a numeric type;
type QUEUING_PORT_NAME_TYPE    is a n-character string;
type QUEUING_DISCIPLINE_TYPE   is (FIFO, PRIORITY);
type MESSAGE_RANGE_TYPE        is a numeric type;
type WAITING_RANGE_TYPE        is a numeric type;
type QUEUING_PORT_STATUS_TYPE  is record
   NB_MESSAGE      : MESSAGE_RANGE_TYPE;       -- current number of messages
   MAX_NB_MESSAGE     : MESSAGE_RANGE_TYPE;
   MAX_MESSAGE_SIZE   : MESSAGE_SIZE_TYPE;
   PORT_DIRECTION     : PORT_DIRECTION_TYPE;
   WAITING_PROCESSES  : WAITING_RANGE_TYPE;
end record;
```

3.6.2 Interpartition Communication Services

The interpartition communication services are divided into two groups, sampling port services and queuing port services.

The sampling port services are:

```
   CREATE_SAMPLING_PORT
   WRITE_SAMPLING_MESSAGE
   READ_SAMPLING_MESSAGE
   GET_SAMPLING_PORT_ID
   GET_SAMPLING_PORT_STATUS
```

The queuing port services are:

```
CREATE_QUEUING_PORT
SEND_QUEUING_MESSAGE
RECEIVE_QUEUING_MESSAGE
GET_QUEUING_PORT_ID
GET_QUEUING_PORT_STATUS
CLEAR_QUEUING_PORT
```

3.6.2.1 Sampling Port Services

A sampling port is a communication object allowing a partition to access a channel of communication configured to operate in sampling mode. In sampling mode, each new occurrence of a message overwrites the previous one. Messages may have a variable length. A refresh period attribute applies to **receive** ports. A validity output parameter indicates whether the age of the read message is consistent with the required refresh period attribute of the port**.** The REFRESH_PERIOD attribute indicates the maximum acceptable age of a valid message, from the time it was received in the port**.**

A sampling port must be created during the **partition** initialization phase before it can be used.

3.6.2.1.1 CREATE_SAMPLING_PORT

The CREATE_SAMPLING_PORT service request is used to create a sampling port. An identifier is assigned by the O/S and returned to the calling process. For a source port, the refresh period is meaningless. At creation, the port is empty.

```
procedure CREATE_SAMPLING_PORT
   (SAMPLING_PORT_NAME   : in   SAMPLING_PORT_NAME_TYPE;
    MAX_MESSAGE_SIZE     : in   MESSAGE_SIZE_TYPE;
    PORT_DIRECTION       : in   PORT_DIRECTION_TYPE;
    REFRESH_PERIOD       : in   SYSTEM_TIME_TYPE;
    SAMPLING_PORT_ID     : out  SAMPLING_PORT_ID_TYPE;
    RETURN_CODE          : out  RETURN_CODE_TYPE) is

error
   when (implementation-defined limit to sampling port creation is exceeded)
      e.g., insufficient storage capacity for the creation of the specified
      port or maximum number of sampling ports have been created =>
      RETURN_CODE := INVALID_CONFIG;
   when (no sampling port of the partition is named SAMPLING_PORT_NAME in the
      configuration tables) =>
      RETURN_CODE := INVALID_CONFIG;
   when (a port named SAMPLING_PORT_NAME is already created) =>
      RETURN_CODE := NO_ACTION;
   when (MAX_MESSAGE_SIZE is zero, negative, or is not equal to
      the value specified in the configuration tables) =>
      RETURN_CODE := INVALID_CONFIG;
   when (PORT_DIRECTION is invalid or is not equal to the
      value specified in the configuration tables) =>
      RETURN_CODE := INVALID_CONFIG;
   when (REFRESH_PERIOD is out of range) =>
      -- e.g., not positive or calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_CONFIG;
   when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

normal
   SAMPLING_PORT_ID := unique identifier assigned by O/S to sampling port named
```

```
      SAMPLING_PORT_NAME;
   RETURN_CODE := NO_ERROR;

end CREATE_SAMPLING_PORT;
```

The return codes for this service are explained below.

CREATE_SAMPLING_PORT

Return Code Value	Commentary
NO_ERROR	Successful completion
NO_ACTION	The port named SAMPLING_PORT_NAME is already created
INVALID_CONFIG	Implementation-defined limit to sampling port creation is exceeded
INVALID_CONFIG	**no sampling port of the partition is named SAMPLING_PORT_NAME in the configuration tables**
INVALID_CONFIG	MAX_MESSAGE_SIZE is **zero, negative,** or is not equal to the value specified in the configuration tables
INVALID_CONFIG	PORT_DIRECTION is invalid or is not equal to the value specified in the configuration tables
INVALID_CONFIG	REFRESH_PERIOD is out of range
INVALID_MODE	Operating mode is NORMAL

3.6.2.1.2 WRITE_SAMPLING_MESSAGE

The WRITE_SAMPLING_MESSAGE service request is used to write a message in the specified sampling port. The message overwrites the previous one.

```
procedure WRITE_SAMPLING_MESSAGE
   (SAMPLING_PORT_ID    : in  SAMPLING_PORT_ID_TYPE;
    MESSAGE_ADDR        : in  MESSAGE_ADDR_TYPE;
    LENGTH              : in  MESSAGE_SIZE_TYPE;
    RETURN_CODE         : out RETURN_CODE_TYPE) is

error
   when (SAMPLING_PORT_ID does not identify an existing sampling port) =>
      RETURN_CODE := INVALID_PARAM;
   when (LENGTH is greater than MAX_MESSAGE_SIZE for the specified port) =>
      RETURN_CODE := INVALID_CONFIG;
   when (LENGTH is zero or negative) =>
      RETURN_CODE := INVALID_PARAM;
   when (the specified port is not configured to operate as a source) =>
      RETURN_CODE := INVALID_MODE;

normal
   write the message represented by MESSAGE_ADDR and LENGTH into the specified port;
   RETURN_CODE := NO_ERROR;

end WRITE_SAMPLING_MESSAGE;
```

The return codes for this service are explained below.

WRITE_SAMPLING_MESSAGE

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	SAMPLING_PORT_ID does not identify an existing sampling port
INVALID_CONFIG	LENGTH is greater than MAX_MESSAGE_SIZE for the specified port
INVALID_PARAM	**LENGTH is zero or negative**
INVALID_MODE	**The specified port** is not configured to operate as a source

3.6.2.1.3 READ_SAMPLING_MESSAGE

The READ_SAMPLING_MESSAGE service request is used to read a message from the specified sampling port. A validity output parameter indicates whether the age of the read message is consistent with the required refresh rate attribute of the port.

COMMENTARY

> It is assumed that, as part of establishing correctness of a message, the core module will not place into the port a message with a length that exceeds the MAX_MESSAGE_SIZE attribute of that port.

```
procedure READ_SAMPLING_MESSAGE
   (SAMPLING_PORT_ID    : in  SAMPLING_PORT_ID_TYPE;
    MESSAGE_ADDR        : in  MESSAGE_ADDR_TYPE;
            -- the message address is passed IN, although the respective message
            -- is passed OUT
    LENGTH              : out MESSAGE_SIZE_TYPE;
    VALIDITY            : out VALIDITY_TYPE;
    RETURN_CODE         : out RETURN_CODE_TYPE) is

error
   when (SAMPLING_PORT_ID does not identify an existing sampling port) =>
      RETURN_CODE := INVALID_PARAM;
   when (the specified port is not configured to operate as a destination) =>
      RETURN_CODE := INVALID_MODE;

normal
   if (sampling port is empty) then
      LENGTH := 0;
      VALIDITY := INVALID;
      RETURN_CODE := NO_ACTION;
   else
      copy last correct message arrived in the specified port to the location
         represented by MESSAGE_ADDR;
      LENGTH := length of the copied message;
      if (age of the copied message is consistent with the required REFRESH_PERIOD
         attribute of the port) then
         VALIDITY := VALID;
      else
         VALIDITY := INVALID;
      end if;
      RETURN_CODE := NO_ERROR;
   end if;
   update validity in status of the port;

end READ_SAMPLING_MESSAGE;
```

The return codes for this service are explained below.

READ_SAMPLING_MESSAGE

Return Code Value	Commentary
NO_ERROR	Successful completion
NO_ACTION	Sampling port is empty
INVALID_PARAM	SAMPLING_PORT_ID does not identify an existing sampling port
INVALID_MODE	**The specified port** is not configured to operate as a destination

3.6.2.1.4 GET_SAMPLING_PORT_ID

The GET_SAMPLING_PORT_ID service **request returns the sampling port identifier that corresponds to** a sampling port name.

```
procedure GET_SAMPLING_PORT_ID
   (SAMPLING_PORT_NAME  : in  SAMPLING_PORT_NAME_TYPE;
    SAMPLING_PORT_ID    : out SAMPLING_PORT_ID_TYPE;
    RETURN_CODE         : out RETURN_CODE_TYPE) is

error
   when (there is no current-partition sampling port named
      SAMPLING_PORT_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

normal
   SAMPLING_PORT_ID := (unique identifier assigned to the sampling port named
      SAMPLING_PORT_NAME)
   RETURN_CODE      := NO_ERROR;

end GET_SAMPLING_PORT_ID;
```

The return codes for this service are explained below.

GET_SAMPLING_PORT_ID

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current-**partition** sampling port named SAMPLING_PORT_NAME

3.6.2.1.5 GET_SAMPLING_PORT_STATUS

The GET_SAMPLING_PORT_STATUS service returns the current status of the specified sampling port.

```
procedure GET_SAMPLING_PORT_STATUS
   (SAMPLING_PORT_ID     : in  SAMPLING_PORT_ID_TYPE;
    SAMPLING_PORT_STATUS : out SAMPLING_PORT_STATUS_TYPE;
    RETURN_CODE          : out RETURN_CODE_TYPE) is

error
   when (SAMPLING_PORT_ID does not identify an existing sampling port) =>
      RETURN_CODE := INVALID_PARAM;

normal
   SAMPLING_PORT_STATUS := current value of port status;
   -- The status should provide the LAST_MSG_VALIDITY that is the validity of
   -- the last read message in the port.
   RETURN_CODE := NO_ERROR;

end GET_SAMPLING_PORT_STATUS;
```

The return codes for this service are explained below.

GET_SAMPLING_PORT_STATUS

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	SAMPLING_PORT_ID does not identify an existing sampling port

3.6.2.2 Queuing Port Services

A queuing port is a communication object allowing a partition to access a channel of communication configured to operate in queuing mode. Messages are stored in FIFO order. Messages may have a variable length. In queuing mode, each new instance of a message cannot overwrite the previous one stored in the send or receive FIFO. However, if the receiving FIFO is full, new messages may be discarded. An appropriate value of **RETURN_CODE (i.e., INVALID_CONFIG) will be returned on the next invocation of the RECEIVE_QUEUING_MESSAGE service**.

A send/respond protocol can be implemented by the applications to guard against communication failures and to ensure flow control between source and destination. The destination partition determines when it will read its messages***.***

A queuing port must be created during the partition initialization phase before it can be used.

3.6.2.2.1 CREATE_QUEUING_PORT

The CREATE_QUEUING_PORT service is used to create a port of communication operating in queuing mode. An identifier is assigned by the O/S and returned to the calling process. At creation, the port is empty. The QUEUING_DISCIPLINE attribute indicates whether blocked processes are queued in FIFO, or in priority order.

```
procedure CREATE_QUEUING_PORT
   (QUEUING_PORT_NAME   : in  QUEUING_PORT_NAME_TYPE;
    MAX_MESSAGE_SIZE    : in  MESSAGE_SIZE_TYPE;
    MAX_NB_MESSAGE      : in  MESSAGE_RANGE_TYPE;
    PORT_DIRECTION      : in  PORT_DIRECTION_TYPE;
    QUEUING_DISCIPLINE  : in  QUEUING_DISCIPLINE_TYPE;
    QUEUING_PORT_ID     : out QUEUING_PORT_ID_TYPE;
    RETURN_CODE         : out RETURN_CODE_TYPE) is
error
   when (implementation-defined limit to queuing port creation is exceeded)
      i.e., Insufficient storage capacity for the creation of the specified
      port or maximum number of queuing ports have been created =>
      RETURN_CODE := INVALID_CONFIG;
   when (no queuing port of the partition is named QUEUING_PORT_NAME in the
      configuration tables) =>
      RETURN_CODE := INVALID_CONFIG;
   when (port named QUEUING_PORT_NAME is already created) =>
      RETURN_CODE := NO_ACTION;
   when (MAX_MESSAGE_SIZE is zero, negative, or is not equal to
      the value specified in the configuration tables) =>
      RETURN_CODE := INVALID_CONFIG;
   when (MAX_NB_MESSAGE is out of range or is not equal to the
      value specified in the configuration tables) =>
      RETURN_CODE := INVALID_CONFIG;
   when (PORT_DIRECTION is invalid or is not equal to the
      value specified in the configuration tables) =>
      RETURN_CODE := INVALID_CONFIG;
   when (QUEUING_DISCIPLINE is invalid) =>
      RETURN_CODE := INVALID_CONFIG;
   when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

normal
   QUEUING_PORT_ID := unique identifier assigned by the O/S to the port named
      QUEUING_PORT_NAME;
   RETURN_CODE     := NO_ERROR;

end CREATE_QUEUING_PORT;
```

The return codes for this service are explained below.

CREATE_QUEUING_PORT

Return Code Value	Commentary
NO_ERROR	Successful completion
NO_ACTION	Port named QUEUING_PORT_NAME is already created
INVALID_CONFIG	Implementation-defined limit to queuing port creation is exceeded
INVALID_CONFIG	**no queuing port of the partition is named QUEUING_PORT_NAME in the configuration tables**
INVALID_CONFIG	MAX_MESSAGE_SIZE is **zero, negative,** or is not equal to the value specified in the configuration tables
INVALID_CONFIG	MAX_NB_MESSAGE is out of range or is not equal to the value specified in the configuration tables
INVALID_CONFIG	PORT_DIRECTION is invalid or is not equal to the value specified in the configuration tables
INVALID_CONFIG	QUEUING_DISCIPLINE is invalid
INVALID_MODE	Operating mode is NORMAL

3.6.2.2.2 SEND_QUEUING_MESSAGE

The SEND_QUEUING_MESSAGE service request is used to send a message in the specified queuing port. If there is sufficient space in the queuing port to accept the message, the message is added to the end of the port’s message queue. If there is insufficient space, the process is blocked and added to the sending process queue, according to the queuing discipline of the port. The process stays on the queue until the specified time-out, if finite, expires or space becomes free in the port to accept the message.

```
procedure SEND_QUEUING_MESSAGE
   (QUEUING_PORT_ID    : in  QUEUING_PORT_ID_TYPE;
    MESSAGE_ADDR       : in  MESSAGE_ADDR_TYPE;
    LENGTH             : in  MESSAGE_SIZE_TYPE;
    TIME_OUT           : in  SYSTEM_TIME_TYPE;
    RETURN_CODE        : out RETURN_CODE_TYPE) is

error
   when (QUEUING_PORT_ID does not identify an existing queuing port) =>
      RETURN_CODE := INVALID_PARAM;
   when (TIME_OUT is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;
   when (LENGTH is greater than MAX_MESSAGE_SIZE for the specified port) =>
      RETURN_CODE := INVALID_CONFIG;
   when (LENGTH is zero or negative) =>
      RETURN_CODE := INVALID_PARAM;
   when (the specified port is not configured to operate as a source port) =>
      RETURN_CODE := INVALID_MODE;

normal
   if (there is sufficient space in the port’s message queue to accept the
      message represented by MESSAGE_ADDR and LENGTH) and (no other process is
      waiting to send a message to that port) then
      insert the message represented by MESSAGE_ADDR and LENGTH in the FIFO
      message queue of the specified port;
      RETURN_CODE := NO_ERROR;
   elsif (TIME_OUT = 0) then
      RETURN_CODE := NOT_AVAILABLE;
   elsif (preemption is disabled or the current process is the error handler
      process) then
      RETURN_CODE := INVALID_MODE;
   else
      if (TIME_OUT is not infinite) then
         initiate a time counter for the current process with duration TIME_OUT;
      end if;
      set the current process state to WAITING;
      insert the process in the port's sending process queue according to the
      queuing discipline of the specified port;
      ask for process scheduling;
      -- The current process is blocked until the expiration of the time-out or
      -- the release of sufficient space in the port's message queue
      if (expiration of the time-out) then
         RETURN_CODE := TIMED_OUT;
      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;
      -- The process is removed from the port's sending process queue and is
      -- set to the READY state (unless another process
      -- suspended it). Process scheduling occurs if preemption is enabled
   end if;

end SEND_QUEUING_MESSAGE;
```

The return codes for this service are explained below.

SEND_QUEUING_MESSAGE Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	QUEUING_PORT_ID does not identify an existing queuing port
INVALID_PARAM	TIME_OUT is out of range
INVALID_CONFIG	LENGTH is greater than MAX_MESSAGE_SIZE for the port
INVALID_PARAM	**LENGTH is zero or negative**
INVALID_MODE	QUEUING_PORT_ID is not configured to operate as a source
NOT_AVAILABLE	Insufficient space in the QUEUING_PORT_ID to accept a new message **and specified TIME_OUT is zero**
INVALID_MODE	(Preemption is disabled or **the current process** is the error handler process) and specified TIME_OUT is not zero
TIMED_OUT	Specified TIME_OUT expired

3.6.2.2.3 RECEIVE_QUEUING_MESSAGE

The RECEIVE_QUEUING_MESSAGE service request is used to receive a message from the specified queuing port. If the queuing port is not empty, the message at the head of the port's message queue is removed and returned to the calling process. If the queuing port is empty, the process is blocked and added to the receiving process queue, according to the queuing discipline of the port. The process stays on the queue until the specified time-out, if finite, expires or a message arrives in the port.

COMMENTARY

If the port’s message queue has overflowed, the message at the head of the port’s message queue is removed and returned to the calling process and the return code **is set to INVALID_CONFIG to** indicate the queue overflow.

It is assumed that, as part of establishing validity of a message, the core module will not place into the port a message with a length that exceeds the MAX_MESSAGE_SIZE attribute of that port.

```
procedure RECEIVE_QUEUING_MESSAGE
   (QUEUING_PORT_ID    : in  QUEUING_PORT_ID_TYPE;
    TIME_OUT           : in  SYSTEM_TIME_TYPE;
    MESSAGE_ADDR       : in MESSAGE_ADDR_TYPE;
            -- the message address is passed IN, although the respective message
            -- is passed OUT
    LENGTH             : out MESSAGE_SIZE_TYPE;
    RETURN_CODE        : out RETURN_CODE_TYPE) is

error
   when (QUEUING_PORT_ID does not identify an existing queuing port) =>
      RETURN_CODE := INVALID_PARAM;
   when (TIME_OUT is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;
   when (the specified port is not configured to operate as a destination
      port) =>
      RETURN_CODE := INVALID_MODE;

normal
   if (the FIFO message queue of the specified port is not empty) then
      copy the first message of the port’s message queue to the
      location represented by MESSAGE_ADDR;
      remove that message from the port’s message queue;
      LENGTH := length of the copied message;
      if (an overflow of the received message queue has occurred) then
          -- the last received messages were not added to the queue
         RETURN_CODE := INVALID_CONFIG;
      else
         RETURN_CODE := NO_ERROR;
      end if;
   elsif (TIME_OUT = 0) then
      LENGTH := 0;
      RETURN_CODE := NOT_AVAILABLE;
   elsif (preemption is disabled or the current process is error handler
      process) then
      LENGTH := 0;
      RETURN_CODE := INVALID_MODE;
   else
      if (TIME_OUT is not infinite) then
         initiate a time counter for the current process with duration TIME_OUT;
      end if;
      set the current process state to WAITING;
      insert the process in the port’s receiving process queue according to the
      queuing discipline of the specified port;
      ask for process scheduling;
      -- The current process is blocked until the expiration of the time-out or
      -- The reception of a new valid message in the specified port
      if (expiration of the time-out) then
```

```
         LENGTH := 0;
         RETURN_CODE := TIMED_OUT;
      else
         -- At the end of a new valid message reception in the specified port
         if (TIME_OUT is not infinite) then
            stop the time counter;
         end if;
         copy the new message from the port’s message queue to
         the location represented by MESSAGE_ADDR;
         remove this new message from the port’s message queue;
         LENGTH := length of the copied message;
         RETURN_CODE := NO_ERROR;
      end if;
      -- The process is removed from the port’s receiving process queue and is
      -- set to the READY state (unless another process
      -- suspended it). Process scheduling occurs if preemption is enabled
   end if;
   -- if no process is waiting in the port’s receiving process queue then
   -- the new message is put in the port’s message queue

end RECEIVE_QUEUING_MESSAGE;
```

The return codes for this service are explained below.

RECEIVE_QUEUING_MESSAGE

Return Code Value	Commentary
NO_ERROR	Successful completion
TIMED_OUT	Specified TIME_OUT expired
INVALID_PARAM	QUEUING_PORT_ID does not identify an existing queuing port
INVALID_PARAM	TIME_OUT is out of range
INVALID_MODE	**The specified port** is not configured to operate as a destination port
INVALID_CONFIG	The queue has overflowed since it was last read
NOT_AVAILABLE	There is no message in the **specified port and the specified TIME_OUT is zero**
INVALID_MODE	(Preemption is disabled or **the current process** is the error handler process*)* and TIME_OUT is not zero

3.6.2.2.4 GET_QUEUING_PORT_ID

The GET_QUEUING_PORT_ID service **request returns the queuing port identifier that corresponds to a queuing port name.**

```
procedure GET_QUEUING_PORT_ID
   (QUEUING_PORT_NAME   : in  QUEUING_PORT_NAME_TYPE;
    QUEUING_PORT_ID     : out QUEUING_PORT_ID_TYPE;
    RETURN_CODE         : out RETURN_CODE_TYPE) is

error
   when (there is no current-partition queuing port named QUEUING_PORT_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

normal
   QUEUING_PORT_ID := (unique identifier assigned to the queuing port named
   QUEUING_PORT_NAME);
   RETURN_CODE     := NO_ERROR;

end GET_QUEUING_PORT_ID;
```

The return codes for this service are explained below.

GET_QUEUING_PORT_ID

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current**-partition** queuing port named QUEUING_PORT_NAME

3.6.2.2.5 GET_QUEUING_PORT_STATUS

The GET_QUEUING_PORT_STATUS service **request** returns the current status of the specified queuing port.

```
procedure GET_QUEUING_PORT_STATUS
   (QUEUING_PORT_ID     : in  QUEUING_PORT_ID_TYPE;
    QUEUING_PORT_STATUS : out QUEUING_PORT_STATUS_TYPE;
    RETURN_CODE         : out RETURN_CODE_TYPE) is

error
   when (QUEUING_PORT_ID does not identify an existing queuing port) =>
      RETURN_CODE := INVALID_PARAM;

normal
   QUEUING_PORT_STATUS := current value of port status;
   RETURN_CODE := NO_ERROR;

end GET_QUEUING_PORT_STATUS;
```

The return codes for this service are explained below.

GET_QUEUING_PORT_STATUS

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	QUEUING_PORT_ID does not identify an existing queuing port

3.6.2.2.6 CLEAR_QUEUING_PORT

The CLEAR_QUEUING_PORT service request discards any messages in the specified port's receive queue.

The service request has no affect on processes waiting on the queuing port (i.e., when there are no messages in the port's receive queuing).

```
procedure CLEAR_QUEUING_PORT
   (QUEUING_PORT_ID    : in  QUEUING_PORT_ID_TYPE;
    RETURN_CODE        : out RETURN_CODE_TYPE) is

error
   when (QUEUING_PORT_ID does not identify an existing queuing port) =>
      RETURN_CODE := INVALID_PARAM;
   when (the specified port is not configured to operate as a
      destination port) =>
      RETURN_CODE := INVALID_MODE;

normal
   empty the contents of the message queue;
   set the number of messages for this port to zero;
   RETURN_CODE := NO_ERROR;

end CLEAR_QUEUING_PORT;
```

The return codes for this service are explained below.

CLEAR_QUEUING_PORT

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	QUEUING_PORT_ID does not identify an existing queuing port
INVALID_MODE	The specified port is not configured as a destination port

3.7 Intrapartition Communication

Two communication mechanisms are available for intrapartition communication. The first **mechanism allows inter-process communication and synchronization via partition-defined** buffers and blackboards. The second **allows inter-process synchronization via partition-defined** counting semaphores and events.

3.7.1 Intrapartition Communication Types

These types are used in intra-partition communication services.

```
type MESSAGE_ADDR_TYPE       is a continuous area of data defined by a starting
                             address;
type MESSAGE_SIZE_TYPE       is a numeric type;
type WAITING_RANGE_TYPE      is a numeric type;
type QUEUING_DISCIPLINE_TYPE is (FIFO, PRIORITY);
```

These types are used in buffer services.

```
type BUFFER_NAME_TYPE   is a n-character string;
type BUFFER_ID_TYPE     is a numeric type;
type MESSAGE_RANGE_TYPE is a numeric type;
type BUFFER_STATUS_TYPE is record
   NB_MESSAGE          : MESSAGE_RANGE_TYPE;
   MAX_NB_MESSAGE      : MESSAGE_RANGE_TYPE;
   MAX_MESSAGE_SIZE    : MESSAGE_SIZE_TYPE;
   WAITING_PROCESSES   : WAITING_RANGE_TYPE;
end record;
```

These types are used in blackboard services.

```
type BLACKBOARD_NAME_TYPE   is a n-character string;
type BLACKBOARD_ID_TYPE     is a numeric type;
type EMPTY_INDICATOR_TYPE   is (EMPTY, OCCUPIED);
type BLACKBOARD_STATUS_TYPE is record
   EMPTY_INDICATOR   : EMPTY_INDICATOR_TYPE;
   MAX_MESSAGE_SIZE  : MESSAGE_SIZE_TYPE;
   WAITING_PROCESSES : WAITING_RANGE_TYPE;
end record;
```

These types are used in semaphore services.

```
type SEMAPHORE_NAME_TYPE    is a n-character string;
type SEMAPHORE_ID_TYPE      is a numeric type;
type SEMAPHORE_VALUE_TYPE   is a numeric type;
type SEMAPHORE_STATUS_TYPE  is record
   CURRENT_VALUE     : SEMAPHORE_VALUE_TYPE;
   MAXIMUM_VALUE     : SEMAPHORE_VALUE_TYPE;
   WAITING_PROCESSES : WAITING_RANGE_TYPE;
end record;
```

These types are used in event services.

```
type EVENT_NAME_TYPE    is a n-character string;
type EVENT_ID_TYPE      is a numeric type;
type EVENT_STATE_TYPE   is (DOWN, UP);
type EVENT_STATUS_TYPE  is record
   EVENT_STATE        : EVENT_STATE_TYPE;
   WAITING_PROCESSES : WAITING_RANGE_TYPE;
end record;
```

The RETURN_CODE_TYPE is common to all APEX services and is defined in Section 3.1.1 of this document.

The SYSTEM_TIME_TYPE is common to many APEX services and is defined in Section 3.4.1 of this document.

3.7.2 Intrapartition Communication Services

The intrapartition communication services are divided into four groups: buffer services, blackboard services, semaphore services, and event services.

The buffer services are:

```
   CREATE_BUFFER
   SEND_BUFFER
   RECEIVE_BUFFER
   GET_BUFFER_ID
   GET_BUFFER_STATUS
```

The blackboard services are:

```
   CREATE_BLACKBOARD
   DISPLAY_BLACKBOARD
   READ_BLACKBOARD
   CLEAR_BLACKBOARD
   GET_BLACKBOARD_ID
   GET_BLACKBOARD_STATUS
```

The semaphore services are:

```
   CREATE_SEMAPHORE
   WAIT_SEMAPHORE
   SIGNAL_SEMAPHORE
   GET_SEMAPHORE_ID
   GET_SEMAPHORE_STATUS
```

The event services are:

```
   CREATE_EVENT
   SET_EVENT
   RESET_EVENT
   WAIT_EVENT
   GET_EVENT_ID
   GET_EVENT_STATUS
```

3.7.2.1 Buffer Services

A buffer is a communication object used by processes of a same partition to send or receive messages. **Multiple messages may be queued in a buffer, but the maximum number of messages supported is defined at buffer creation.** Within a buffer, the messages are queued in FIFO order. The buffer message **size may be variable, but the maximum size is defined** at buffer creation.

A buffer must be created during the **partition** initialization phase before it can be used. A name is given at buffer creation, this name is local to the partition and is not an attribute of the partition configuration table. **The creation of buffers (e.g., names used, number of buffers) for one partition has no impact on the creation of buffers for other partitions.**

3.7.2.1.1 CREATE_BUFFER

The CREATE_BUFFER service request is used to create a message buffer with a maximum number of maximum size messages. A BUFFER_ID is assigned by the O/S and returned to the calling process. Processes can create as many buffers as the pre-allocated memory space will support. The QUEUING_DISCIPLINE input parameter indicates the process queuing policy (FIFO or priority order) associated with that buffer.

```
procedure CREATE_BUFFER
   (BUFFER_NAME          : in  BUFFER_NAME_TYPE;
    MAX_MESSAGE_SIZE     : in  MESSAGE_SIZE_TYPE;
    MAX_NB_MESSAGE       : in  MESSAGE_RANGE_TYPE;
    QUEUING_DISCIPLINE   : in  QUEUING_DISCIPLINE_TYPE;
    BUFFER_ID            : out BUFFER_ID_TYPE;
    RETURN_CODE          : out RETURN_CODE_TYPE) is

error
   when (there is not enough available storage space for the creation of the
      specified buffer or maximum number of buffers have been created) =>
      RETURN_CODE := INVALID_CONFIG;
   when (the buffer named BUFFER_NAME has already been created) =>
      RETURN_CODE := NO_ACTION;
   when (MAX_MESSAGE_SIZE is zero or negative) =>
      RETURN_CODE := INVALID_PARAM;
   when (MAX_NB_MESSAGE is out of range) =>
      RETURN_CODE := INVALID_PARAM;
   when (QUEUING_DISCIPLINE is invalid) =>
      RETURN_CODE := INVALID_PARAM;
   when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

normal
   BUFFER_ID := unique identifier assigned by the O/S to an unallocated buffer
      object;
    set the buffer's process queuing discipline to QUEUING_DISCIPLINE;
    -- This information will be then used by the O/S to order the waiting
    -- processes (FIFO or priority order)
    RETURN_CODE := NO_ERROR;

end CREATE_BUFFER;
```

The return codes for this service are explained below.

CREATE_BUFFER

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is not enough available storage space for the creation of the specified buffer
NO_ACTION	The buffer named BUFFER_NAME has already been created
INVALID_PARAM	MAX_MESSAGE_SIZE is **zero or negative**
INVALID_PARAM	MAX_NB_MESSAGE is out of range
INVALID_PARAM	QUEUING_DISCIPLINE is invalid
INVALID_MODE	Operating mode is NORMAL

3.7.2.1.2 SEND_BUFFER

The SEND_BUFFER service request is used to send a message in the specified buffer. The calling process will be queued while the buffer is full for a maximum duration of the specified time-out.

```
procedure SEND_BUFFER
   (BUFFER_ID    : in  BUFFER_ID_TYPE;
    MESSAGE_ADDR : in  MESSAGE_ADDR_TYPE;
    LENGTH       : in  MESSAGE_SIZE_TYPE;
    TIME_OUT     : in  SYSTEM_TIME_TYPE;
    RETURN_CODE  : out RETURN_CODE_TYPE) is

error
   when (BUFFER_ID does not identify an existing buffer) =>
      RETURN_CODE := INVALID_PARAM;
   when (LENGTH is greater than the MAX_MESSAGE_SIZE specified for the
      buffer) =>
      RETURN_CODE := INVALID_PARAM;
   when (LENGTH is zero or negative) =>
      RETURN_CODE := INVALID_PARAM;
   when (TIME_OUT is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;

normal
   if (message buffer not full) then
      if (no processes are waiting on an empty buffer) then
         store the message in the FIFO message queue of the specified buffer;
      else
         remove the first receiving process from the buffer’s process queue;
         copy the message represented by MESSAGE_ADDR and LENGTH into the
            receiving process’s message area associated with the RECEIVE_BUFFER
            service request made by this receiving process;
         if (this receiving process is waiting on an empty buffer with a
            time-out) then
            stop the time counter associated with the receiving process;
         end if;
         move receiving process from the WAITING to the READY state (unless
            another process suspended it);
         if (preemption is enabled) then
            ask for process scheduling;
            -- The current process may be preempted
         end if;
      end if;
      RETURN_CODE := NO_ERROR;
   elsif (TIME_OUT = 0) then
      RETURN_CODE := NOT_AVAILABLE;
   elsif (preemption is disabled or the current process is the error handler
      process) then
      RETURN_CODE := INVALID_MODE;
   elsif (TIME_OUT is infinite) then
      set the current process state to WAITING;
      insert this process in the buffer’s process queue at the position
      specified by the queuing discipline;
      ask for process scheduling;
      -- The current process is blocked and will be removed
      -- from the buffer’s process queue and return in READY state
      -- by a RECEIVE_BUFFER service request on the buffer from
      -- another process (unless another process suspended it)
      RETURN_CODE := NO_ERROR;
      -- The message represented by MESSAGE_ADDR and LENGTH will be put in the
      -- FIFO message queue by the next RECEIVE service request
   else
      set the process state to WAITING;
      insert this process in the buffer's process queue at the position
         specified by the queuing discipline;
      initiate a time counter for the current process with duration TIME_OUT;
      ask for process scheduling;
      -- The current process is blocked and will be removed
      -- from the buffer's process queue and return in READY state
      -- by expiration of the time-out or by a RECEIVE_BUFFER service
      -- request on the buffer from another process (unless another process
      -- suspended it)
      if (expiration of the time-out) then
         RETURN_CODE := TIMED_OUT;
      else
         RETURN_CODE := NO_ERROR;
         -- The time counter will be stopped and the message will be put in
         -- the FIFO message queue by the next RECEIVE service request
      end if;
   end if;
end SEND_BUFFER;
```

The return codes for this service are explained below.

SEND_BUFFER	
Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	BUFFER_ID does not identify an existing buffer
INVALID_PARAM	LENGTH is greater than MAX_MESSAGE_SIZE specified for the buffer
INVALID_PARAM	LENGTH is zero or negative
INVALID_PARAM	TIME_OUT is out of range
INVALID_MODE	(Preemption is disabled or the current process is the error handler process) and TIME_OUT is not zero
NOT_AVAILABLE	No place in the buffer to put a message
TIMED_OUT	The specified TIME_OUT is expired

3.7.2.1.3 RECEIVE_BUFFER

The RECEIVE_BUFFER service request is used to receive a message from the specified buffer. The calling process will be queued while the buffer is empty for a time-out maximum duration.

```
procedure RECEIVE_BUFFER
   (BUFFER_ID    : in  BUFFER_ID_TYPE;
    TIME_OUT     : in  SYSTEM_TIME_TYPE;
    MESSAGE_ADDR : in MESSAGE_ADDR_TYPE;
            -- the message address is passed IN, although the respective message
            -- is passed OUT
    LENGTH       : out MESSAGE_SIZE_TYPE;
    RETURN_CODE  : out RETURN_CODE_TYPE) is

error
   when (BUFFER_ID does not identify an existing buffer) =>
      RETURN_CODE := INVALID_PARAM;
   when (TIME_OUT is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;

normal
   if (message buffer is not empty) then
      copy the first message of the specified buffer message queue to the
      location represented by MESSAGE_ADDR;
      LENGTH  := Length of the copied message;
      if (there are sending processes waiting on this buffer) then
         -- The buffer was previously full but now has space for another message
         remove the first sending process from the process queue;
         put the message associated with this sending process in the FIFO
         message queue;
         if (this sending process was waiting on the buffer with a time-out)
            then
            stop the affected time counter;
         end if;
         move this sending process from the WAITING to the READY state (unless
            another process suspended it);
         if (preemption is enabled) then
            ask for process scheduling;
            -- The current process may be preempted
         end if;
      end if;
      RETURN_CODE := NO_ERROR;
   elsif (TIME_OUT = 0) then
      LENGTH := 0;
      RETURN_CODE := NOT_AVAILABLE;
   elsif (preemption is disabled or the current process is the error handler
      process) then
      LENGTH := 0;
      RETURN_CODE := INVALID_MODE;
   elsif (TIME_OUT is infinite) then
      set the current process state to WAITING;
      insert the current process in the buffer's process queue at the position
      specified by the queuing discipline;
      ask for process scheduling;
      -- The current process is blocked and will be removed from
      -- the buffer's process queue and return in
      -- READY state by a SEND_BUFFER service request on the buffer from
      -- another process (unless another process suspended it)
      RETURN_CODE := NO_ERROR;
      -- The next SEND_BUFFER service request will copy the sent message into
      -- the location represented by MESSAGE_ADDR and LENGTH
   else
      set the current process state to WAITING;
      insert the current process in the buffer's process queue at the position
      specified by the queuing discipline;
      initiate a time counter for the current process with duration TIME_OUT;
      ask for process scheduling;
      -- The current process is blocked and will be removed from the
      -- buffer's process queue and return in READY state by expiration
      -- of the time-out or by a SEND_BUFFER service request on the
      -- buffer from another process (unless another process suspended it)
      if (expiration of the time-out) then
         LENGTH := 0;
         RETURN_CODE := TIMED_OUT;
      else
         RETURN_CODE := NO_ERROR;
         -- The next SEND_BUFFER service request will stop the time counter and
         -- will copy the sent message into the location represented by
         -- MESSAGE_ADDR and set LENGTH to the length of the copied message
      end if;
   end if;
end RECEIVE_BUFFER;
```

The return codes for this service are explained below.

RECEIVE_BUFFER

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	BUFFER_ID does not identify an existing buffer
INVALID_PARAM	TIME_OUT is out of range
INVALID_MODE	(Preemption is disabled or current process is error handler process) and TIME_OUT is not zero
NOT_AVAILABLE	The buffer does not contain any message
TIMED_OUT	The specified TIME_OUT expired

3.7.2.1.4 GET_BUFFER_ID

The GET_BUFFER_ID service request **returns the buffer identifier that corresponds to a buffer** name.

```
procedure GET_BUFFER_ID
   (BUFFER_NAME   : in  BUFFER_NAME_TYPE;
    BUFFER_ID     : out BUFFER_ID_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (the current partition has no buffer named BUFFER_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

normal
   BUFFER_ID   := unique identifier assigned to the buffer named BUFFER_NAME;
   RETURN_CODE := NO_ERROR;

end GET_BUFFER_ID;
```

The return codes for this service are explained below.

GET_BUFFER_ID

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current-partition buffer named BUFFER_NAME

3.7.2.1.5 GET_BUFFER_STATUS

The GET_BUFFER_STATUS service request returns the status of the specified buffer.

```
procedure GET_BUFFER_STATUS
   (BUFFER_ID     : in  BUFFER_ID_TYPE;
    BUFFER_STATUS : out BUFFER_STATUS_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (BUFFER_ID does not identify an existing buffer) =>
      RETURN_CODE := INVALID_PARAM;

normal
   BUFFER_STATUS :=
      (NB_MESSAGE         => current number of messages inside the buffer,
       MAX_NB_MESSAGE     => buffer’s maximum number of messages,
       MAX_MESSAGE_SIZE   => buffer’s maximum size of messages,
       WAITING_PROCESSES => the number of processes waiting on the buffer);
   RETURN_CODE := NO_ERROR;

end GET_BUFFER_STATUS;
```

The return codes for this service are explained below.

```
GET_BUFFER_STATUS
Return Code Value        Commentary
NO_ERROR                 Successful completion
INVALID_PARAM            BUFFER_ID does not identify an existing buffer
```

3.7.2.2 Blackboard Services

A blackboard is a communication object used by processes of a same partition to send or receive messages. A blackboard does not use message queues, each new occurrence of a message overwrites **any other. The blackboard message size may be variable, but the maximum size is defined at blackboard creation.**

A blackboard must be created during the **partition** initialization phase before it can be used. The memory size given in the configuration table should include the memory size necessary to manage all the blackboards of a partition. A name is given at blackboard creation, this name is local to the partition and is not an attribute of the partition configuration table. **The creation of blackboards (e.g., names used, number of blackboards) for one partition has no impact on the creation of blackboards for other partitions.**

COMMENTARY

> A process is put in waiting state in case of a read request on an empty blackboard. This is a main difference with the interpartition sampling mode.

3.7.2.2.1 CREATE_BLACKBOARD

The CREATE_BLACKBOARD service request is used to create a blackboard with a specified **maximum** message size. A BLACKBOARD_ID is assigned by the O/S and returned to the calling process. Processes can create as many buffers as the pre-allocated memory space will support.

```
procedure CREATE_BLACKBOARD
   (BLACKBOARD_NAME   : in  BLACKBOARD_NAME_TYPE;
    MAX_MESSAGE_SIZE : in  MESSAGE_SIZE_TYPE;
    BLACKBOARD_ID    : out BLACKBOARD_ID_TYPE;
    RETURN_CODE      : out RETURN_CODE_TYPE) is

error
   when (there is not enough available storage space for the creation of the
      specified blackboard or maximum number of blackboards has been
      created) =>
      RETURN_CODE   := INVALID_CONFIG;
   when (the blackboard named BLACKBOARD_NAME has already been created) =>
      RETURN_CODE   := NO_ACTION;
   when (MAX_MESSAGE_SIZE is zero or negative) =>
      RETURN_CODE   := INVALID_PARAM;
   when (operating mode is NORMAL) =>
      RETURN_CODE   := INVALID_MODE;

normal
   BLACKBOARD_ID := (unique identifier of an unallocated blackboard object);
   set the blackboard's empty indicator to EMPTY;
   RETURN_CODE := NO_ERROR;

end CREATE_BLACKBOARD;
```

The return codes for this service are explained below.

CREATE_BLACKBOARD

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is not enough available storage space for the creation of the specified blackboard **or maximum number of blackboards has been created**
NO_ACTION	The blackboard named BLACKBOARD_NAME has already been created
INVALID_PARAM	MAX_MESSAGE_SIZE is **zero or negative**
INVALID_MODE	Operating mode is NORMAL

3.7.2.2.2 DISPLAY_BLACKBOARD

The DISPLAY_BLACKBOARD service request is used to display a message in the specified blackboard. The specified blackboard becomes not empty. **If processes were waiting on the empty blackboard, the processes will be released on a priority followed by FIFO (when priorities are equal) basis.**

```
procedure DISPLAY_BLACKBOARD
   (BLACKBOARD_ID : in  BLACKBOARD_ID_TYPE;
    MESSAGE_ADDR  : in  MESSAGE_ADDR_TYPE;
    LENGTH        : in  MESSAGE_SIZE_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (BLACKBOARD_ID does not identify an existing blackboard) =>
      RETURN_CODE := INVALID_PARAM;
   when (LENGTH is greater than MAX_MESSAGE_SIZE specified for the
      blackboard) =>
      RETURN_CODE := INVALID_PARAM;
   when (LENGTH is zero or negative) =>
      RETURN_CODE := INVALID_PARAM;

normal
   set the blackboard's empty indicator to OCCUPIED;
   overwrite the contents of the specified blackboard with the message
      represented by MESSAGE_ADDR and LENGTH;
   if (there are processes waiting on an empty blackboard) then
      if (some of them are waiting with a time-out) then
         stop the affected time counters;
      end if;
      remove the processes from the blackboard's process queue and move them
         from the WAITING to the READY state (unless another process
         suspended a waiting process);
      if (preemption is enabled) then
         ask for process scheduling;
         -- The current process may be preempted
      end if;
   end if;
   RETURN_CODE := NO_ERROR;

end DISPLAY_BLACKBOARD;
```

The return codes for this service are explained below.

DISPLAY_BLACKBOARD

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	BLACKBOARD_ID does not identify an existing blackboard

INVALID_PARAM	**LENGTH is greater than MAX_MESSAGE_SIZE specified for the blackboard**
INVALID_PARAM	**LENGTH is zero or negative**

3.7.2.2.3 READ_BLACKBOARD

The READ_BLACKBOARD service request is used to read a message in the specified blackboard. The calling process will be in waiting state while the blackboard is empty for **up to the specified TIME_OUT** duration.

There is no process queuing discipline attribute associated with blackboards. When processes are waiting on an empty blackboard and another process writes a message on that blackboard, then all the processes waiting on that blackboard become ready and will read the last available message on that blackboard in any case. **A process waiting on a blackboard will never be returned to the READY state and receive an empty blackboard message (i.e., the last written message is always available).**

```
procedure READ_BLACKBOARD
   (BLACKBOARD_ID : in  BLACKBOARD_ID_TYPE;
    TIME_OUT      : in  SYSTEM_TIME_TYPE;
    MESSAGE_ADDR  : in MESSAGE_ADDR_TYPE;
            -- the message address is passed IN, although the respective message
            -- is passed OUT
    LENGTH        : out MESSAGE_SIZE_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (BLACKBOARD_ID does not identify an existing blackboard) =>
      RETURN_CODE := INVALID_PARAM;
   when (TIME_OUT is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;

normal
   if (empty indicator of the specified blackboard is OCCUPIED) then
      copy the message displayed in the specified blackboard
         to the location represented by MESSAGE_ADDR;
      LENGTH  := Length of the copied message;
      RETURN_CODE := NO_ERROR;
   elsif (TIME_OUT = 0) then
      LENGTH := 0;
      RETURN_CODE := NOT_AVAILABLE;
   elsif (preemption is disabled or the current process is the error handler
      process) then
      LENGTH := 0;
      RETURN_CODE := INVALID_MODE;
   elsif (TIME_OUT is infinite) then
      set the current process state to WAITING;
      insert the process in the blackboard's process queue;
      ask for process scheduling;
      -- The current process is blocked and will be removed from the
      -- blackboard's process queue and return in READY state by a
      -- DISPLAY_BLACKBOARD service request on the blackboard from another
      -- process (unless another process suspended it)
      copy the last available message of the blackboard to the
         location represented by MESSAGE_ADDR;  -- Even if the message has been
                                                -- cleared by the
                                                -- CLEAR_BLACKBOARD service.
      LENGTH  := Length of the copied message;
      RETURN_CODE := NO_ERROR;
   else
      set the current process state to WAITING;
      insert the process in the blackboard’s process queue;
      initiate a time counter for the current process with duration TIME_OUT;
      ask for process scheduling;
      -- The current process is blocked and will be removed from the
      -- blackboard’s process queue and return in READY state by
      -- expiration of the time-out or by a DISPLAY_BLACKBOARD service request
      -- on the blackboard from another process (unless another process
      -- suspended it)
      if (expiration of the time-out) then
         LENGTH := 0;
        RETURN_CODE := TIMED_OUT;
      else
         copy the last available blackboard message to the location
            represented by MESSAGE_ADDR;
         LENGTH  := Length of the copied message;
         RETURN_CODE := NO_ERROR;
         -- The next DISPLAY_BLACKBOARD service request will stop the time
         -- counter
      end if;
   end if;

end READ_BLACKBOARD;
```

The return codes for this service are explained below.

READ_BLACKBOARD

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	BLACKBOARD_ID does not identify an existing blackboard
INVALID_PARAM	TIME_OUT is out of range
INVALID_MODE	(Preemption is disabled or the current process is the error handler process) and TIME_OUT is not zero
NOT_AVAILABLE	No message in the blackboard
TIMED_OUT	The specified TIME_OUT expired

3.7.2.2.4 CLEAR_BLACKBOARD

The CLEAR_BLACKBOARD service request is used to clear the message of the specified blackboard. The specified blackboard becomes empty.

This service request has no affect on processes waiting on the blackboard (i.e., when the blackboard is already empty).

```
procedure CLEAR_BLACKBOARD
   (BLACKBOARD_ID : in  BLACKBOARD_ID_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (BLACKBOARD_ID does not identify an existing blackboard) =>
      RETURN_CODE := INVALID_PARAM;

normal
   Set the empty indicator of the specified blackboard to EMPTY;
   RETURN_CODE := NO_ERROR;

end CLEAR_BLACKBOARD;
```

The return codes for this service are explained below.

CLEAR_BLACKBOARD

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	BLACKBOARD_ID does not identify an existing blackboard

3.7.2.2.5 GET_BLACKBOARD_ID

The GET_BLACKBOARD_ID service request **returns the blackboard identifier that corresponds to a blackboard name.**

```
procedure GET_BLACKBOARD_ID
   (BLACKBOARD_NAME : in  BLACKBOARD_NAME_TYPE;
    BLACKBOARD_ID   : out BLACKBOARD_ID_TYPE;
    RETURN_CODE     : out RETURN_CODE_TYPE) is

error
   when (the current partition has no blackboard named BLACKBOARD_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

normal
   BLACKBOARD_ID := unique identifier assigned to the blackboard named
      BLACKBOARD_NAME;
   RETURN_CODE   := NO_ERROR;

end GET_BLACKBOARD_ID;
```

The return codes for this service are explained below.

GET_BLACKBOARD_ID

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current-partition blackboard named BLACKBOARD_NAME

3.7.2.2.6 GET_BLACKBOARD_STATUS

The GET_BLACKBOARD_STATUS service request returns the status of the specified blackboard.

```
procedure GET_BLACKBOARD_STATUS
   (BLACKBOARD_ID     : in  BLACKBOARD_ID_TYPE;
    BLACKBOARD_STATUS : out BLACKBOARD_STATUS_TYPE;
    RETURN_CODE       : out RETURN_CODE_TYPE) is

error
   when (BLACKBOARD_ID does not identify an existing blackboard) =>
      RETURN_CODE := INVALID_PARAM;

normal
   BLACKBOARD_STATUS :=
      (EMPTY_INDICATOR   => the value of the blackboard's empty indicator,
       MAX_MESSAGE_SIZE  => the blackboard's configured maximum message size,
       WAITING_PROCESSES => the number of processes waiting on the blackboard);
   RETURN_CODE := NO_ERROR;

end GET_BLACKBOARD_STATUS;
```

The return codes for this service are explained below.

GET_BLACKBOARD_STATUS

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	BLACKBOARD_ID does not identify an existing blackboard

3.7.2.3 Semaphore Services

Counting semaphore services are supported. A counting semaphore is a synchronization object commonly used to provide access to partition resources.

A semaphore must be created during the **partition's** initialization phase before it can be used. A name is given at semaphore creation, this name is local to the partition and is not an attribute of the partition configuration table. **The creation of semaphores (e.g., names used, number of semaphores) for one partition has no impact on the creation of semaphores for other partitions.**

3.7.2.3.1 CREATE_SEMAPHORE

The CREATE_SEMAPHORE service request is used to create a semaphore of a specified current and maximum value. The maximum value parameter is the maximum value that the semaphore can be signaled to **(e.g., number of resources managed by the semaphore)**. The current value is the semaphores' starting value after creation. For example, if the semaphore was used to manage access to five resources, and at the time of creation three resources were available, the semaphore would be created with a maximum value of five and a current value of three. The QUEUING_DISCIPLINE input parameter indicates the process queuing policy (FIFO or priority order) associated with the semaphore.

```
procedure CREATE_SEMAPHORE
   (SEMAPHORE_NAME       : in   SEMAPHORE_NAME_TYPE;
    CURRENT_VALUE        : in   SEMAPHORE_VALUE_TYPE;
    MAXIMUM_VALUE        : in   SEMAPHORE_VALUE_TYPE;
    QUEUING_DISCIPLINE   : in   QUEUING_DISCIPLINE_TYPE;
    SEMAPHORE_ID         : out  SEMAPHORE_ID_TYPE;
    RETURN_CODE          : out  RETURN_CODE_TYPE) is

error
   when (the maximum number of semaphores has been created) =>
      RETURN_CODE  := INVALID_CONFIG;
   when (a semaphore named SEMAPHORE_NAME has already been created) =>
      RETURN_CODE  := NO_ACTION;
   when (CURRENT_VALUE is out of range) =>
      RETURN_CODE  := INVALID_PARAM;
   when (MAXIMUM_VALUE is out of range) =>
      RETURN_CODE  := INVALID_PARAM;
   when (CURRENT_VALUE is greater than MAXIMUM_VALUE) =>
      RETURN_CODE  := INVALID_PARAM;
   when (QUEUING_DISCIPLINE is invalid) =>
      RETURN_CODE  := INVALID_PARAM;
   when (operating mode is NORMAL) =>
      RETURN_CODE  := INVALID_MODE;

normal
   SEMAPHORE_ID :=  unique identifier assigned by the O/S to an unallocated
      semaphore object;
   set the semaphore's process queuing discipline to QUEUING_DISCIPLINE;
   -- This information will be then used by the O/S to order processes in the
   -- semaphore's process queue(FIFO or priority order)
   RETURN_CODE := NO_ERROR;
   initialize semaphore with MAXIMUM_VALUE and CURRENT_VALUE;

end CREATE_SEMAPHORE;
```

The return codes for this service are explained below.

CREATE_SEMAPHORE Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	The maximum number of semaphores has been created
NO_ACTION	The semaphore named SEMAPHORE_NAME has already been created
INVALID_PARAM	CURRENT_VALUE is out of range
INVALID_PARAM	MAXIMUM_VALUE is out of range
INVALID_PARAM	CURRENT_VALUE **is greater than** MAXIMUM_VALUE
INVALID_PARAM	QUEUING_DISCIPLINE is invalid
INVALID_MODE	Operating mode is NORMAL

3.7.2.3.2 WAIT_SEMAPHORE

The WAIT_SEMAPHORE service request **is used to attempt to acquire (i.e., decrements) one count of a semaphore's value (e.g., request for one resource managed by the semaphore). The service** moves the current process from the running state to the waiting state if the current value of the specified semaphore is zero and if the specified time-out is not zero. **If the current value of the specified semaphore is positive,** the process goes on executing and the semaphore's current value is decremented.

```
procedure WAIT_SEMAPHORE
   (SEMAPHORE_ID  : in  SEMAPHORE_ID_TYPE;
    TIME_OUT      : in  SYSTEM_TIME_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (SEMAPHORE_ID does not identify an existing semaphore) =>
      RETURN_CODE := INVALID_PARAM;
   when (TIME_OUT is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;

normal
   if (current value of the specified semaphore is greater than 0) then
      decrement current value of the specified semaphore;
      RETURN_CODE := NO_ERROR;
   elsif (TIME_OUT = 0) then
      RETURN_CODE := NOT_AVAILABLE;
   elsif (preemption is disabled or the current process is the error handler
      process) then
      RETURN_CODE := INVALID_MODE;
   elsif (TIME_OUT is infinite) then
      set the current process state to WAITING;
      insert the current process in the semaphore's process queue at the
         position specified by the queuing discipline;
      ask for process scheduling;
      -- The current process is blocked and will be removed from the
      -- semaphore's process queue and return in READY state by a
      -- SIGNAL_SEMAPHORE service request on the semaphore from another
      -- process (unless another process
      -- suspended it)
      RETURN_CODE := NO_ERROR;
   else
      -- ((TIME_OUT > 0) and (Current value = 0))
      set the current process state to WAITING;
      insert the current process in the semaphore’s process queue at the
         position specified by the queuing discipline;
      initiate a time counter for the current process with duration TIME_OUT;
      ask for process scheduling;
      -- The current process is blocked and will be removed from
      -- the semaphore’s process queue and
      -- return in READY state by expiration of the TIME_OUT or by a
      -- SIGNAL_SEMAPHORE service request on the semaphore from another
      -- process (unless another process suspended it)
      if (expiration of the time-out) then
         RETURN_CODE := TIMED_OUT;
      else
         RETURN_CODE := NO_ERROR;
         -- The next SIGNAL_SEMAPHORE service request will stop the time counter
      end if;
   end if;

end WAIT_SEMAPHORE;
```

The return codes for this service are explained below.

WAIT_SEMAPHORE

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	SEMAPHORE_ID does not identify an existing semaphore
INVALID_PARAM	TIME_OUT is out of range
INVALID_MODE	(Preemption is disabled or the current process is the error handler process) and TIME_OUT is not zero
NOT_AVAILABLE	Current value of SEMAPHORE_ID<=0 and TIME_OUT=0
TIMED_OUT	The specified TIME_OUT expired

3.7.2.3.3 SIGNAL_SEMAPHORE

If the semaphore’s value is not equal to its maximum, the SIGNAL_SEMAPHORE service request increments the current value of the specified semaphore **(e.g., the current process completed its usage of one resource managed by the semaphore)**. If processes are waiting on that semaphore, the first process of the queue is moved from the waiting state to the ready state and a scheduling takes place.

```
procedure SIGNAL_SEMAPHORE
   (SEMAPHORE_ID     : in  SEMAPHORE_ID_TYPE;
    RETURN_CODE      : out RETURN_CODE_TYPE) is

error
   when (SEMAPHORE_ID does not identify an existing semaphore) =>
      RETURN_CODE := INVALID_PARAM;
   when (the semaphore’s current value equals its maximum value) =>
      RETURN_CODE := NO_ACTION;

normal
   if (no processes are waiting on the specified semaphore) then
      increment the current value of the specified semaphore;
      RETURN_CODE := NO_ERROR;
   else
      remove the first process from the semaphore’s process queue;
      if (this process is waiting on the semaphore with a time-out) then
         stop the affected time counter;
      end if;
      move this process from the WAITING to the READY state (unless
         another process suspended it);
      if (preemption is enabled) then
         ask for process scheduling;
         -- The current process may be preempted
      end if;
      RETURN_CODE := NO_ERROR;
   end if;

end SIGNAL_SEMAPHORE;
```

The return codes for this service are explained below.

SIGNAL_SEMAPHORE

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	SEMAPHORE_ID does not identify an existing semaphore
NO_ACTION	**The semaphore's current value equals its** maximum value

3.7.2.3.4 GET_SEMAPHORE_ID

The GET_SEMAPHORE_ID service request **returns the semaphore identifier that corresponds to a semaphore name.**

```
procedure GET_SEMAPHORE_ID
   (SEMAPHORE_NAME : in  SEMAPHORE_NAME_TYPE;
    SEMAPHORE_ID   : out SEMAPHORE_ID_TYPE;
    RETURN_CODE    : out RETURN_CODE_TYPE) is

error
   when (there is no current-partition semaphore named SEMAPHORE_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

normal
   SEMAPHORE_ID := unique identifier assigned to the semaphore named
      SEMAPHORE_NAME;
   RETURN_CODE  := NO_ERROR;

end GET_SEMAPHORE_ID;
```

The return codes for this service are explained below.

GET_SEMAPHORE_ID

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current-partition semaphore named SEMAPHORE_NAME

3.7.2.3.5 GET_SEMAPHORE_STATUS

The GET_SEMAPHORE_STATUS service request returns the status of the specified semaphore.

```
procedure GET_SEMAPHORE_STATUS
   (SEMAPHORE_ID     : in  SEMAPHORE_ID_TYPE;
    SEMAPHORE_STATUS : out SEMAPHORE_STATUS_TYPE;
    RETURN_CODE      : out RETURN_CODE_TYPE) is

error
   when (SEMAPHORE_ID does not identify an existing semaphore) =>
      RETURN_CODE := INVALID_PARAM;

normal
   SEMAPHORE_STATUS :=
      (CURRENT_VALUE     => Current value of the specified semaphore,
       MAXIMUM_VALUE     => Maximum value of the specified semaphore,
       WAITING_PROCESSES => the number of processes waiting on the semaphore);
   RETURN_CODE := NO_ERROR;

end GET_SEMAPHORE_STATUS;
```

The return codes for this service are explained below.

GET_SEMAPHORE_STATUS

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	SEMAPHORE_ID does not identify an existing semaphore

3.7.2.4 Event Services

An event is a synchronization object used to notify the occurrence of a condition to processes that **may be waiting for condition to occur.**

An event must be created during the **partition's** initialization phase before it can be used. A name is given at event creation, this name is local to the partition and is not an attribute of the partition configuration table. **The creation of events (e.g., names used, number of events) for one partition has no impact on the creation of events for other partitions.**

3.7.2.4.1 CREATE_EVENT

The CREATE_EVENT service request creates an event object for use by any of the process in the partition. Upon creation the event is set to the down state.

```
procedure CREATE_EVENT
   (EVENT_NAME  : in  EVENT_NAME_TYPE;
    EVENT_ID    : out EVENT_ID_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (the maximum number of events has been created) =>
      RETURN_CODE := INVALID_CONFIG;
   when (the event named EVENT_NAME has already been created) =>
      RETURN_CODE := NO_ACTION;
   when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

normal
   EVENT_ID := unique identifier assigned by the O/S to an unallocated event
      object;
   set the event's state to the DOWN state;
   RETURN_CODE := NO_ERROR;

end CREATE_EVENT;
```

The return codes for this service are explained below.

CREATE_EVENT

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	The maximum number of events has been created
NO_ACTION	The event named EVENT_NAME has already been created
INVALID_MODE	Operating mode is NORMAL

3.7.2.4.2 SET_EVENT

The SET_EVENT service request sets the specified event to the up state. All the processes waiting on the event are moved from the waiting state to the ready state **(unless suspended while**

waiting on the event), and a scheduling takes place. **If processes were waiting on the event, the processes will be released on a priority followed by FIFO (when priorities are equal) basis.**

```
procedure SET_EVENT
   (EVENT_ID    : in  EVENT_ID_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (EVENT_ID does not identify an existing event) =>
      RETURN_CODE := INVALID_PARAM;

normal
   set the specified event to the UP state;
   if (there are any processes waiting for that event) then
      if (some of them are waiting  with a time-out) then
         stop the affected time counters;
      end if;
      remove each waiting process from the event’s process queue and move them
         from the WAITING state to READY state (unless
         another process suspended a waiting process);
      if (preemption is enabled) then
         ask for process scheduling;
         -- The current process may be preempted by a process that was waiting
         -- on the event
      end if;
      RETURN_CODE := NO_ERROR;
   end if;

end SET_EVENT;
```

The return codes for this service are explained below.

SET_EVENT

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	EVENT_ID does not identify an existing event

3.7.2.4.3 RESET_EVENT

The RESET_EVENT service request sets the specified event in down state.

```
procedure RESET_EVENT
   (EVENT_ID    : in  EVENT_ID_TYPE;
    RETURN_CODE : out RETURN_CODE_TYPE) is

error
   when (EVENT_ID does not identify an existing event) =>
      RETURN_CODE := INVALID_PARAM;

normal
   set the specified event to the DOWN state;
   RETURN_CODE := NO_ERROR;

end RESET_EVENT;
```

The return codes for this service are explained below.

RESET_EVENT

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	EVENT_ID does not identify an existing event

3.7.2.4.4 WAIT_EVENT

The WAIT_EVENT service request moves the current process from the running state to the waiting state if the specified event is down and if the specified time-out is not zero. The process goes on executing if the specified event is up or if it is a conditional wait (event down and time-out is zero).

COMMENTARY

> There is no process queuing discipline attribute associated with events. When processes are waiting on an event and another process sets the event to up, then all the processes waiting on the event become ready.

```
procedure WAIT_EVENT
   (EVENT_ID          : in  EVENT_ID_TYPE;
    TIME_OUT          : in  SYSTEM_TIME_TYPE;
    RETURN_CODE       : out RETURN_CODE_TYPE) is

error
   when (EVENT_ID does not identify an existing event) =>
      RETURN_CODE := INVALID_PARAM;
   when (TIME_OUT is out of range) =>
      -- e.g., calculation causes overflow of underlying clock
      RETURN_CODE := INVALID_PARAM;

normal
   if (the state of the specified event is UP) then
      RETURN_CODE := NO_ERROR;
   elsif (TIME_OUT = 0) then
      RETURN_CODE := NOT_AVAILABLE;
   elsif (preemption is disabled or the current process is the error handler
      process) then
      RETURN_CODE := INVALID_MODE;
   elsif (TIME_OUT is infinite) then
      set the current process state to WAITING;
      insert the current process in the event's process queue;
      ask for process scheduling;
      -- The current process is blocked and will be removed from the
      -- event's process queue and return in READY state
      -- by a SET_EVENT service request on the event from another process
      -- (unless another process suspended it)
      RETURN_CODE := NO_ERROR;
   else
      -- ((TIME_OUT) > 0 and (event is DOWN))
      set the current process state to WAITING;
      insert the current process in the event's process queue;
      initiate a time counter for the current process with duration TIME_OUT;
      ask for process scheduling;
      -- The current process is blocked and will be removed from the
      -- event's process queue and return in READY state
      -- by expiration of the time-out or by a
      -- SET_EVENT service request from another process (unless another
      -- process suspended it)
      if (expiration of the time-out) then
         RETURN_CODE := TIMED_OUT;
      else
         RETURN_CODE := NO_ERROR;
         -- The next SET_EVENT service request will stop the time counter
      end if;
   end if;

end WAIT_EVENT;
```

The return codes for this service are explained below.

WAIT_EVENT

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	EVENT_ID does not identify an existing event
INVALID_PARAM	TIME_OUT is out of range
INVALID_MODE	(Preemption is disabled or **the current process** is the error handler process) and TIME_OUT is not zero
NOT_AVAILABLE	The event is in the DOWN state
TIMED_OUT	The specified TIME_OUT expired

3.7.2.4.5 GET_EVENT_ID

The GET_EVENT_ID **service request returns the event identifier that corresponds to an event name.**

```
procedure GET_EVENT_ID
   (EVENT_NAME    : in  EVENT_NAME_TYPE;
    EVENT_ID      : out EVENT_ID_TYPE;
    RETURN_CODE   : out RETURN_CODE_TYPE) is

error
   when (there is no current-partition event named EVENT_NAME) =>
      RETURN_CODE := INVALID_CONFIG;

normal
   EVENT_ID := unique identifier assigned to the event named EVENT_NAME;
   RETURN_CODE := NO_ERROR;

end GET_EVENT_ID;
```

The return codes for this service are explained below.

GET_EVENT_ID

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	There is no current-partition event named EVENT_NAME

3.7.2.4.6 GET_EVENT_STATUS

The GET_EVENT_STATUS service request returns the status of the specified event.

```
procedure GET_EVENT_STATUS
   (EVENT_ID     : in  EVENT_ID_TYPE;
    EVENT_STATUS : out EVENT_STATUS_TYPE;
    RETURN_CODE  : out RETURN_CODE_TYPE) is

error
   when (EVENT_ID does not identify an existing event) =>
      RETURN_CODE := INVALID_PARAM;

normal
   EVENT_STATUS :=
      (EVENT_STATE       => state of the specified event,
       WAITING_PROCESSES => the number of processes waiting on the specified
      event);
   RETURN_CODE := NO_ERROR;
```

```
end GET_EVENT_STATUS;
```

The return codes for this service are explained below.

GET_EVENT_STATUS

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	EVENT_ID does not identify an existing event

3.8 Health Monitoring

The Health Monitor (HM) is invoked by an application calling the RAISE_APPLICATION_ERROR service or by the O/S or hardware detecting a fault. The recovery action is dependent on the error level (see Section 2.4.2.1). The recovery actions (see Section 2.4.2.2) for each module and each partition level error are specified in the HM tables. The recovery actions for process level errors are defined by the application programmer in a special error handler process.

3.8.1 Health Monitoring Types

```
MAX_ERROR_MESSAGE_SIZE is fixed in the appendix of the standard.

type SYSTEM_ADDRESS_TYPE      is language dependent;
type STACK_SIZE_TYPE          is a numeric type;
type ERROR_MESSAGE_TYPE       is a continuous area of data of
                              MAX_ERROR_MESSAGE_SIZE;
type ERROR_MESSAGE_SIZE_TYPE  is a numeric type;

type ERROR_CODE_TYPE is   --implementation independent enumeration
(DEADLINE_MISSED,
 APPLICATION_ERROR,
 NUMERIC_ERROR,
 ILLEGAL_REQUEST,
 STACK_OVERFLOW,
 MEMORY_VIOLATION,
 HARDWARE_FAULT,
 POWER_FAIL);

type ERROR_STATUS_TYPE is record
   ERROR_CODE        : ERROR_CODE_TYPE;
   MESSAGE           : ERROR_MESSAGE_TYPE;
   LENGTH            : ERROR_MESSAGE_SIZE_TYPE;
   FAILED_PROCESS_ID : PROCESS_ID_TYPE;
   FAILED_ADDRESS    : SYSTEM_ADDRESS_TYPE;
end record;
```

COMMENTARY

> **The contents of FAILED_ADDRESS is implementation dependent. It may represent an address associated with the current process or another process (e.g., due to deadline miss). It may represent an address associated with an instruction or a memory location.**

The RETURN_CODE_TYPE is common to all APEX services **and is defined in Section 3.1.1 of this document**.

3.8.2 Health Monitoring Services

The Health Monitoring services are:

```
REPORT_APPLICATION_MESSAGE
CREATE_ERROR_HANDLER
GET_ERROR_STATUS
RAISE_APPLICATION_ERROR
```

3.8.2.1 REPORT_APPLICATION_MESSAGE

The REPORT_APPLICATION_MESSAGE service request allows the current partition to transmit a message to the HM function. REPORT_APPLICATION_MESSAGE may be used to record an event for logging purposes.

```
procedure REPORT_APPLICATION_MESSAGE
   (MESSAGE_ADDR : in  MESSAGE_ADDR_TYPE;
    LENGTH       : in  MESSAGE_SIZE_TYPE;
    RETURN_CODE  : out RETURN_CODE_TYPE) is

error
   when (LENGTH is out of range) =>
      RETURN_CODE := INVALID_PARAM;

normal
   transmit the message represented by MESSAGE_ADDR and LENGTH to the Health
      Monitoring function;
   RETURN_CODE := NO_ERROR;

end REPORT_APPLICATION_MESSAGE;
```

The return codes for this service are explained below.

REPORT_APPLICATION_MESSAGE

Return Code Value	Commentary
INVALID_PARAM	LENGTH is out of range
NO_ERROR	Successful completion

3.8.2.2 CREATE_ERROR_HANDLER

The CREATE_ERROR_HANDLER service request creates an error handler process for the current partition. This process has no identifier (ID) and cannot be accessed by the other processes of the partition. The error handler is a special aperiodic process with the highest priority, **no deadline, and whose entry point defined by this service is invoked by the O/S when a process level error (e.g., deadline missed, numeric error, stack overflow) is detected. It is executed during the partition's time windows**. It cannot be suspended nor stopped by another process. Its priority cannot be modified. It preempts any running process regardless of its priority and even if preemption is disabled (i.e., lock level not equal to zero). **When the error handler is ready to be finished, it invokes** the STOP_SELF service.

If the error handler is not created, the recovery action taken is as specified in the Partition HM table.

The error handler is written by the application programmer. It can stop and restart the failed process with the STOP and START services, restart (COLD_START or WARM_START) the entire partition by the SET_PARTITION_MODE (COLD_START or WARM_START) service, or shut down the partition by the SET_PARTITION_MODE (IDLE) service. But, the error handler should not be used to correct an error (e.g., limit a value in case of overflow). The faulty process can continue its execution only in the case of application error or deadline missed.

The error handler process cannot call blocking services. If any code running in the context of the error handler calls LOCK_PREEMPTION or UNLOCK_PREEMPTION no action will be taken. This is because the error handler is already the highest priority process and cannot be interrupted or blocked. It can transmit the error context to the HM function via the REPORT_APPLICATION_MESSAGE service for maintenance purpose.

The identifier of the faulty process, the error code, the error address, and fault message can be obtained by using the GET_ERROR_STATUS service. The error handler process is responsible for reporting the fault.

In case of a fault detected when the error handler process is running (e.g., memory violation, O/S fault), the error handler process cannot be safely executed in the partition context. The recovery action specified by the Partition HM table is automatically applied.

```
procedure CREATE_ERROR_HANDLER
   (ENTRY_POINT      : in  SYSTEM_ADDRESS_TYPE;
    STACK_SIZE       : in  STACK_SIZE_TYPE;
    RETURN_CODE      : out RETURN_CODE_TYPE) is

error
   when (error handler process is already created) =>
      RETURN_CODE := NO_ACTION;
   when (insufficient storage capacity for the creation of the error
      handler process) =>
      RETURN_CODE := INVALID_CONFIG;
   when (STACK_SIZE is out of range) =>
      RETURN_CODE := INVALID_CONFIG;
   when (operating mode is NORMAL) =>
      RETURN_CODE := INVALID_MODE;

normal
   Create a special process (i.e., no process ID) with the highest priority,
   ENTRY_POINT and STACK_SIZE attributes;
   -- The ENTRY_POINT will be invoked by the O/S    -- when a process level
                                                    -- error is detected
   RETURN_CODE := NO_ERROR;

end CREATE_ERROR_HANDLER;
```

The return codes for this service are explained below.

CREATE_ERROR_HANDLER

Return Code Value	Commentary
NO_ERROR	Successful completion
NO_ACTION	Error handler is already created
INVALID_CONFIG	insufficient storage capacity for the creation of the error handler process
INVALID_CONFIG	**STACK_SIZE** is out of range
INVALID_MODE	Operating mode is NORMAL

3.8.2.3 GET_ERROR_STATUS

The GET_ERROR_STATUS service must be used by the error handler process to determine the error code, the identifier of the faulty process, the address at which the error occurs, and the message associated with the fault.

If more than one process is **in error, this service must be called in a loop in the error handler until there are no more processes are reported as being in error.**

COMMENTARY

> The errors are stored temporarily in a FIFO queue by the HM when they occur and are removed by the GET_ERROR_STATUS service requests in the FIFO order. If the error was raised by the application with the RAISE_APPLICATION_ERROR, the message is the one provided by the application; otherwise, for the other errors, the message is implementation dependent.

```
procedure GET_ERROR_STATUS
   (ERROR_STATUS     : out ERROR_STATUS_TYPE;
    RETURN_CODE      : out RETURN_CODE_TYPE) is

error
   when (the current process is not the error handler) =>
      RETURN_CODE := INVALID_CONFIG;
   when (there is no process in error) =>
      RETURN_CODE := NO_ACTION;

normal
   ERROR_STATUS := error status of the first process in the process error list;
   clear this error from the process error list;
   -- the error handler process cannot be restarted for the same error
   -- occurrence
   RETURN_CODE := NO_ERROR;

end GET_ERROR_STATUS;
```

The return codes for this service are explained below.

GET_ERROR_STATUS

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_CONFIG	The current process is not the error handler
NO_ACTION	There is no process in error

3.8.2.4 RAISE_APPLICATION_ERROR

The RAISE_APPLICATION_ERROR service request allows the current partition to invoke the error handler process for the specific error code APPLICATION_ERROR. The message passed will be read with the GET_ERROR_STATUS. The error handler of the partition is then started (if created) to take the recovery action for the process which raises the error code; otherwise (the error handler is not created) the error is considered at partition level error.

If the RAISE_APPLICATION_ERROR service is requested by the error handler process, this error is considered a partition level error. In this case, the recovery action for an application error specified by the partition HM table of the current partition is automatically applied.

COMMENTARY

> The ERROR_CODE parameter has been retained for backwards compatibility. It must always be set to APPLICATION_ERROR.

```
procedure RAISE_APPLICATION_ERROR
   (ERROR_CODE   : in  ERROR_CODE_TYPE;
    MESSAGE_ADDR : in  MESSAGE_ADDR_TYPE;
    LENGTH       : in  ERROR_MESSAGE_SIZE_TYPE;
    RETURN_CODE  : out RETURN_CODE_TYPE) is
```

```
error
   when (LENGTH is negative or is greater than MAX_ERROR_MESSAGE_SIZE) =>
      RETURN_CODE := INVALID_PARAM;
   when (ERROR_CODE is not APPLICATION_ERROR) =>
      RETURN_CODE := INVALID_PARAM;

normal
   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
      start the error handler process for the error code in the current process;
      ask for process scheduling;
      -- the current process will be preempted by error handler even if
      -- preemption is disabled;
      -- the input parameters are made available to GET_ERROR_STATUS;
   end if;
   RETURN_CODE := NO_ERROR;

end RAISE_APPLICATION_ERROR;
```

The return codes for this service are explained below.

RAISE_APPLICATION_ERROR

Return Code Value	Commentary
NO_ERROR	Successful completion
INVALID_PARAM	LENGTH is negative or greater than MAX_ERROR_MESSAGE_SIZE
INVALID_PARAM	ERROR_CODE is not APPLICATION_ERROR

4.1 Compliance to APEX Interface

This Specification provides requirements that an O/S implementor should implement and conform to.

The O/S interface should provide a “guarantee” to the application supplier. It describes system services and data structures with specified semantics that can be relied upon if the application runs on a conforming O/S.

Therefore, there are two types of conformance to this standard, implementation conformance and application conformance.

4.2 O/S Implementation Compliance

An O/S conforming to ARINC 653 should support all required system services and data structures, including the functional behavior described in this standard. The O/S implementer should use a compliance test suite based on the “Conformity Test Specification” provided by ARINC 653 Part 3. The conformity test specification, including PASS/FAIL criteria, is intended to be independent of implementation. Passing the conformity test suite successfully will demonstrate that the candidate is conforming to the standards defined in ARINC 653 Part 1.

A conforming implementation may support additional services or data objects. It may even implement non-standard extensions or additional APIs. However, it should define an environment in which an application can be run with the API and functional behavior described in this standard.

If an O/S does not support the entire set of ARINC 653 required system services, any limitations should be clearly documented. The extent to which optional features are implemented and their possible impact on safety should also be clearly documented.

4.3 Application Compliance

A conforming application can require only the O/S facilities described in this standard. For unspecified or implementation-defined behavior, a conforming application should accept any behavior implemented by a conforming O/S.

5.1 XML Configuration Specification

This section defines the mechanism for specifying configuration data, in a standard format, that does not depend on the implementation of the ARINC 653 **core software**. The intent is to provide a concise method for defining the configuration in a manner that is not dependent on a particular implementation. This provides an intermediate form of the configuration definition that allows the system integrator to create configuration specifications in a form that can be readily converted into an implementation-specific configuration. It is not intended that XML be **used "as is" by the core software**. The implementer may choose to do this, but it is not required.

The ARINC 653 XML-Schema types define the structures and the types of the data needed to configure any ARINC 653 O/S. These XML-Schema types are used by the ARINC 653 O/S implementer to create the XML-Schema that is specific to the O/S implementation. The XML-Schema is used by the system integrator to write the XML configuration.

The ARINC 653 XML-Schema types cannot be altered by the O/S implementer. They may be extended by including them in larger types, if necessary, to incorporate extra configuration types that are specific to the O/S implementation. An example of how ARINC 653 XML-schema types can be extended is provided in Appendix I.

COMMENTARY

> **For portability of applications across different implementations, O/S implementers should avoid custom configuration elements, except as indicated otherwise in this standard.**

The ARINC 653 XML-Schema types are defined in Appendix H and are implemented in distinct files to clearly identify the types that can be extended from the ones that cannot:

- **ARINC653Types.xsd: this file contains most of the ARINC653 XML types and should not be altered by the O/S implementer.**
- **ARINC653TypeExtensible.xsd: contains simple types that can be extended and is included by ARINC653Types.xsd. These simple types list possible choices that may be expended by adding new entries in the list.**

From one implementation to another, the system integrator should find the same ARINC 653 XML-Schema types, making it easier to transfer configuration data.

It is expected that an ARINC 653 compliant operating system should be able to be configured using the required configuration data defined in the XML schema types. In doing so, the module should function in a manner that is compliant with this standard. The XML-schema type definition covers only the configuration of the ARINC 653 compliant core module. The integrated module and higher-level systems may require additional configuration data (e.g. sensor/actuator configuration, data bus communications configuration, etc.), which are out of scope of this standard. It is not the intent of this standard to restrict the use, or definition of parameters, which are outside of the ARINC 653 compliant core module.

A Graphical representation of the XML-Schema types is provided in Appendix G. An example of an XML schema and its instance file is shown in Appendix I.

The following section explains how the types are used to produce an XML Schema and how this schema is used to produce and verify an XML configuration instance.

5.2 XML Workflow Example

This section describes a typical workflow necessary to develop an XML-Schema using the ARINC 653 XML-Schema types, how this XML-Schema is used for producing an instance file, and then translated to the O/S implementation-specific format. This example has six major steps for defining and using the XML-Schema shown graphically in Figure 5-1. The steps are described below.

1. **The ARINC 653 working group standardizes ARINC 653 XML-Schema types. They are defined in Attachment H and implemented in two files as mentioned above.**
2. The ARINC 653 core O/S implementer **will develop its own XML-Schema by completing the extensible section of the schema.**
3. **The O/S implementer develops a translator tool. The translator tool uses the ARINC 653 XML-Schema as the reference for converting an XML instance file into the configuration format used by the core O/S implementation.**
4. The system integrator and/or the application developer define the XML instance file for the configuration being integrated. This XML instance **must comply with the XML-Schema created by the core O/S implementer in Step 2.**
5. The system Integrator and/or the application developer validate the XML instance file. A validation tool can be used to verify the instance file is well formed and valid against the XML-Schema.
6. The system integrator uses the core O/S implementer's translator to convert the XML instance file into the configuration format used by the core O/S implementation.

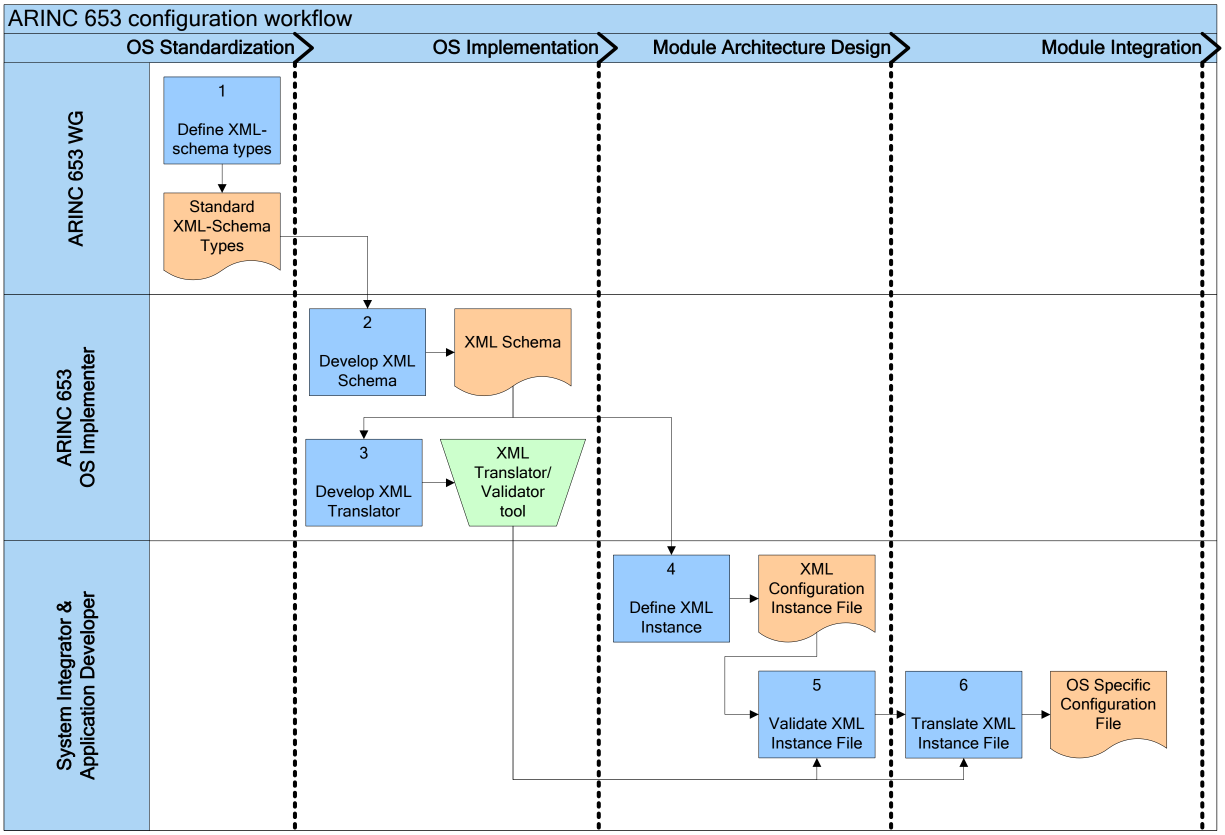

ARINC 653 configuration workflow
OS Standardization
OS Implementation
Module Architecture Design
Module Integration
ARINC 653 WG
ARINC 653 OS Implementer
System Integrator & Application Developer
1 Define XML-schema types
Standard XML-Schema Types
2 Develop XML Schema
XML Schema
3 Develop XML Translator
XML Translator/ Validator tool
4 Define XML Instance
XML Configuration Instance File
5 Validate XML Instance File
6 Translate XML Instance File
OS Specific Configuration File

Figure 5–1 Defining and Using XML-Schema and XML Instance File

APPENDIX A
GLOSSARY

Acknowledgement
Indication to the sender of a message that the message has arrived at its destination and has been recognized.

Ada
High level programming language developed as a standard by the US DOD. Ada83 became an ISO standard in 1987.

Ada83
The original Ada standard defined by the 1983 Ada Language Reference Manual (LRM).

Ada95
The revised Ada standard defined by the 1995 Ada Language Reference Manual (LRM).

Ada Task
Programming unit of the Ada language. A task may be scheduled concurrently with other tasks (i.e., has its own thread of execution).

Address Space
A range of consecutive locations addressable by the processor. An address space can be physical (directly represents the underlying hardware) or virtual (indirectly represents the underlying hardware by using different address values to correspond to the physical address space).

Algorithm
A finite set of well-defined rules that give a sequence of operations for performing a specific task. (DO-178B).

Aperiodic
Occurs predictably but not at regular time intervals.

Application
That software consisting of tasks or processes that perform a specific function on the aircraft. An application may be composed of one or more partitions (ARINC Report 651, RTCA/DO-255).

Application Partition
An ARINC 653 compliant partition (further defined in 1.3.2a, first occurrence).

Backplane
The physical circuit card and components consisting of the electrical connection points for interfacing cabinet resources to the outside world and integrating avionics modules. (ARINC Report 651)

Backplane Bus
A data transfer bus (serial or parallel) used for inter-module communications that is associated with the backplane that core modules are plugged into.

APPENDIX A
GLOSSARY

Baseline
The approved, documented configuration of one or more configuration items, that thereafter serves as the basis for further development and that can be changed only through change control procedures. (DO-178B)

Broadcast
Message Transmission from a single source to all available destinations.

Build
An operational version of a software product incorporating a specific subset of capabilities that the final product will include. (ARINC Report 652)

Blocked Process
A process that cannot continue execution, generally due to not being able to obtain a resource (e.g., message, semaphore, event) required to continue execution.

Cabinet
The physical structure used in IMA to provide an environmental barrier and house the avionics modules, cabinet resources, and avionics backplane. (ARINC Report 651)

Certification
The process of obtaining regulatory agency approval for a function, equipment, system, or aircraft by establishing that it complies with all applicable government regulations. (ARINC 652) See also DO-178B.

Channel
A path for interpartition communications, consists of a set of logically connected ports.

Compiler
Program that translates source code statements in a high order language, such as Ada or C, into object code. (DO-178B)

Compliance
Demonstrable adherence to a set of mandatory requirements.

Component
A self-contained part, combination of parts, subassemblies or units, which performs a distinct function of a system. (DO-178B)

Configuration Tables
Data structures that contain ARINC 653 configuration settings for a module.

Conformance
Providing all services and characteristics described in a specification or standard.

Conformance Test
Test program which demonstrates that required services have been provided and characteristics exist.

APPENDIX A
GLOSSARY

Core Hardware

A hardware environment (e.g., processor, memory, I/O resources, etc.) used in a core module.

Core Module

A hardware environment (Core Hardware) and any accompanying software components (Core Software) capable to host a single ARINC 653 API execution context (i.e., execute a single partition schedule at any given time).

Core Software

Software components (e.g., O/S, hardware interface software, etc.) of a core module.

Criticality Level

The degree to which loss or malfunction of a system will affect aircraft performance.

Critical Software

Software implemented in or related to aircraft performance or safety. (ARINC Report 652)

Cyclic

Actions which occur in a fixed repeated order but not necessarily at fixed time intervals.

Database

A set of data, part or the whole of another set of data, consisting of at least one file that is sufficient for a given purpose or for a given data processing system. (DO-178B)

Deadline

A time by which a process must have completed a certain activity.

Debug

The process of locating, analyzing, and correcting suspected errors. (ARINC Report 652)

Default

A value or state which is used when contrary values or actions are not available.

Deterministic

The ability to produce a predictable outcome generally based on the preceding operations. The outcome occurs in a specified **elapsed** time with some degree of repeatability.

Dormant

A process which is available to execute but is not currently executing or waiting to execute. (See text of ARINC Specification 653)

Error

1. With respect to software, a mistake in requirements, design or code. (DO-178B)
2. An undesired system state that exists either at the boundary or at an internal point in the system; it may be experienced by the user as failure when it is manifested at the boundary. For software, it is the programmer action or omission that results in a fault.

Executable Object Code

The binary form of code instructions which is directly usable by the CPU.

Executive

See Operating System.

Failure

The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered. (IEEE Std 729-1983) (DO-178B)

Fault

A manifestation of an error in software. A fault, if encountered, may cause a failure. (IEEE Std 729-1983) (DO-178B)

Fault Avoidance

Minimizing the occurrence of faults. (ARINC Report 652)

Faults, Common Mode

Coincident faults resulting from an error present in several identical redundant hardware or software components; typically generic in nature, “designed-in fault.” (ARINC Report 652)

Fault Containment

To ensure that all faults are isolated to an individual LRU/LRM for maintenance action. (ARINC Report 652)

Fault Containment Region

Hardware/software/system partitioning which ensures that errors or failures do not propagate to other non-faulty partitions. (ARINC Report 652)

Fault Detection

The ability to positively identify that a failure has occurred (i.e., a fault was triggered). (ARINC Report 652)

Faults, Generic

Faults resulting from requirements, specification, design, implementation, or operational/support errors or deficiencies. Note that by this definition, software faults must be classified as generic. Also, a “designed-in” fault not attributed to uncontrollable environmental factors. (ARINC Report 652)

Fault Isolation

The ability of a system to identify the location of a fault once a failure has occurred. (ARINC Report 652)

Fault Masking

Computational technique to permit correct system function in the presence of faults without requiring identification of the faulty valve. (ARINC Report 652)

Fault Tolerance

The built-in capability of a system to provide continued correct execution in the presence of a limited number of hardware or software faults. (ARINC Report 652)

Fault Tolerance Renewal

Returning a fault tolerant component to a completely operational state. (ARINC Report 652)

APPENDIX A
GLOSSARY

FIFO (First In First Out) Queue

Queuing system where entries are processed in the order in which they were placed on the queue.

Hardware/Software Integration

The process of combining the software into the target computer.

High Order Language

A programming language that usually includes features such as nested expressions, user defined data types, and parameter passing normally not found in lower order languages, that does not reflect the structure of any one given computer or class of computers, and that can be used to write machine independent source programs. A single, higher-order language statement may represent multiple machine operations. (ARINC Report 652)

Implementation Dependent

A feature or setting that may vary between implementations (i.e., vary between O/S vendors).

Independence

Separation of responsibilities which assures the accomplishment of objective evaluation and the authority to ensure corrective action.

Initialization

A sequence of actions which bring the system to a state of operational readiness.

Installer

The group or organization for product installation, system integration and certification of a product on the aircraft. Usually the aircraft manufacturer, although it may be a separate installation contractor, or sometimes an element of the user’s organization. (ARINC Report 652)

Integrated Module

A core module and all partitions it hosts. Also referred to as Module.

Interchangeability

When either the current element or its replacement satisfies the same functional and interface requirements.

Interpartition

Refers to any communication conducted between partitions.

Inter-module Communications

Mechanism used to communicate between core modules (e.g., backplane bus).

Interrupt

A suspension of a task, such as the execution of a computer program, caused by an event external to that task and performed in such a way that the task can be resumed. (ARINC Report 652)

Intrapartition

Refers to any communication conducted within a partition.

Linker

A program which assembles individual modules of object code, which reference each other into a single module of executable object code.

Loader

A routine that reads an object program into main storage prior to its execution. (ARINC Report 652)

Major Time Frame

A time interval containing a sequence of partition time windows allocated to partitions within a module, which is periodically repeated throughout the module's runtime operation.

Message

A package of data transmitted between partitions, or between partitions and external entities. Messages are sent and received by partitions via sampling and queuing port services.

Module

Refer to the definition of Integrated Module.

Module HM Table

The Module HM configuration table defines the HM behavior for errors that occur outside a partition window (i.e., during the module initialization, or during a partition switch).

Multi-Partition HM Table

The Multi-Partition HM configuration table defines the global HM behavior for errors that occur within a partition window. There is exactly one Multi-Partition HM Table associated with each partition; however, the same Multi-Partition HM Table may be used by a set of partitions.

Object

In this standard, an object is a software data structure concept associated with the create services. In order to interact with the O/S, an object related to the ARINC 653 mechanism (e.g., process, sampling port, semaphore, etc.) is allocated as part of the create service call.

Operating System (O/S)

(deleted by Supplement 1)

Partition

A program, including instruction code and data, that is loadable into a single address space in a module. The operating system has absolute control over a partition's use of processing time, memory, and other resources such that each partition is isolated from all others sharing the core module.

Partitioning

(deleted by Supplement 1)

Partition Duration

The amount of execution time required by the partition within one partition period.

APPENDIX A
GLOSSARY

Partition HM Table

The Partition HM configuration table defines the local HM behavior for errors that occur inside partition windows of a given partition.

Partition Period

The required time interval at which the partition must be activated in order to satisfy its operational requirements.

Partition Time Window

An uninterrupted interval of execution time provided to a partition within a partition schedule. The interval is defined by Partition Time Window Duration and Partition Time Window Offset.

Partition Time Window Duration

The quantity of execution time in the partition time window.

Partition Time Window Offset

Time between the start of the major time frame and the activation of the partition time window.

Period

For a periodic process, the period of activation of the process. For a partition, refer to the definition of Partition Period.

Periodic

Occurs cyclically with a fixed time period.

Periodic Processing Start

A point in a partition schedule aligned to the beginning of a partition’s window where a given partition’s periodic process scheduling is permitted to start.

Port

A partition defined resource for sending or receiving messages over a specific channel. The attributes of a port define the message requirements and characteristics needed to control transmissions.

Preemptive

The executing process may be suspended to allow a higher priority process to execute.

Priority Based Scheduling

Scheduling where the highest priority process, in the ready to run state, is executed.

Priority Inversion

A process assumes a higher or lower priority than it is allocated. This may arise as follows: during execution, process P initiates process P1 which has a higher priority. P is therefore suspended awaiting the result of P1. Meanwhile process P2 becomes ready to execute. P2 has a priority higher than P but lower than P1. P2 is therefore effectively blocked by process P which has a lower priority.

Priority Queue

Queuing system where entries which are assigned highest priority by their originators are processed first.

Process

A programming unit contained within a partition which executes concurrently with other processes of the same partition. A process is the same as a task. (ARINC Report 651)

Processor

A device used for processing digital data. (ARINC Report 651)

Queue

A queue is a concept used within the ARINC 653 specification as a means to track a set of related items (e.g., process queue, message queue). Items are added and removed from the queue in an order based on attributes associated with the queue and the data items (e.g., priority, first-in-first out).

Redundancy

Standard fault-tolerance technique, detailed below:

NMR: N-modular redundancy--critical system components/modules are replicated N-times, with voting or some type of acceptance test used to make consistent decisions and detect disagreement.

TMR: Triple modular redundancy--critical system components/modules are included in triplicate, with voting or some type of acceptance test used to detect disagreement and make decisions (version of N-modular redundancy with N = 3). (ARINC Report 652)

Robust Partitioning

A mechanism for assuring the intended isolation of independent aircraft operational functions residing in shared computing resources in all circumstances, including hardware and programming errors. The objective of Robust Partitioning is to provide the same level of functional isolation as a federated implementation (i.e., applications individually residing on separate computing elements). This means robust partitioning must support the cooperative coexistence of applications on a**n integrated module**, while assuring unauthorized, or unintended interference is prevented. Robust partitioning should comply with the following guidance provided in RTCA DO-248B/EUROCAE ED-94B:

A software partition should not be allowed to contaminate another partition’s code, I/O, or data storage areas.

A software partition should be allowed to consume shared processor resources only during its **allocated time windows**.

A software partition should be allowed to consume shared I/O resources only during its **allocated time windows**.

Failures of hardware unique to a software partition should not cause adverse effects on other software partitions.

Software providing partitioning should have the same or higher software level than the highest level of the partitioned software applications.

APPENDIX A
GLOSSARY

Run-Time

An integrated module's main operational state where executable object code is running on the target computer and applications are performing their intended functions.

Segmentation

Division of a message into smaller units (segments) to enable transmission.

Stack

Area of memory, allocated to a process, utilized on a last in first out basis (LIFO).

Standard Partition

From a module's perspective, a partition that is within the module.

Suspended

Process in waiting state. Execution has been temporarily halted awaiting completion of another activity or occurrence of an event.

System Build Time

Time period that represents life cycle activities related to building the system (e.g., generating executable object code, generating object programs to be used by the loader). These activities occur before and outside of the operational times (e.g., initialization time, run time).

System Partition

A partition that requires interfaces outside the ARINC 653 defined services, but is still constrained by robust spatial and temporal partitioning. A system partition may perform functions such as managing communication from hardware devices or fault management schemes. System partitions are optional and are specific to the core module implementation.

System Integrator

The organization **that, among other activities,** configures **and integrates** the applications onto an IMA infrastructure.

Task

A programming unit contained within a partition which executes concurrently with other processes of the same partition. A task is the same as a process. (ARINC Report 651)

Terminated

Process in dormant state. Execution has ended and cannot be resumed.

Voting

A technique used to combine redundant inputs/outputs/computations; voting types include majority, exact agreement (bit-by-bit), and approximate agreement. Used for fault masking, where the goal is achieving valid results from inputs, not all of which are non-faulty. (ARINC Report 652)

APPENDIX B
ACRONYMS

ACR	Avionics Computer Resource
AEEC	Airlines Electronic Engineering Committee
AJPO	Ada Joint Program Office
APEX	APplication EXecutive
API	Application Program Interface
ARTEWG	Ada Runtime Environment Working Group of SIGAda
ARTS	Ada Runtime System
BITE	Built In Test Equipment
CARID	Catalog of Ada Runtime Implementation Dependencies
CIFO	Catalog of Interface Functions and Options for Ada Runtime Environment
COTS	Commercial Off The Shelf – applies to commercially available software or hardware intended to satisfy avionic system software functional requirements.
ExTRA	Extensions For Real Time Ada
FIFO	First In/First Out
HM	Health Monitor
HOL	Higher-Order Language
IEEE	Institute of Electrical and Electronic Engineers
IMA	Integrated Modular Avionics
I/O	Input/Output
ISO	International Organization for Standardization (See OSI)
LEX	Lexical Analyzer
LRU	Line Replaceable Unit
MMU	Memory Management Unit
MOPS	Minimum Operational Standard
MRTSI	Model Runtime System Interface
O/S	Operating System
OMS	Onboard Maintenance System
OSI	Open System Interconnection - An ISO standard communications model (See ISO)
POSIX	Portable Operating System Interface Standard

APPENDIX C
APEX SERVICES SPECIFICATION GRAMMAR

(This Appendix deleted by Supplement 3.)

ADA 83 INTERFACE SPECIFICATION

```
-- This is a compilable Ada Specification of the APEX interface, compatible
-- with Ada 83 or Ada 95, and derived from Section 3 of ARINC 653 Part 1.
--
-- The declarations of the services given below are taken from the
-- standard, as are the enumerated types and the names of the others types.
-- Unless specified as implementation dependent, the values specified in
-- this appendix should be implemented as defined.
--
-- All types have a defined representation clause to enable compatibility
-- between the C and Ada 95 specifications. The objective is to have
-- the same definitions and representations of the interface in both the Ada
-- and C languages. The specifications are aligned in the same order, and
-- define the same items.


-- ---------------------------------------------------------------------------
--
-- Global constant and type definitions
--
-- ---------------------------------------------------------------------------

with SYSTEM;

package APEX_TYPES is

   -- ---------------------------
   -- Domain limits            --
   -- ---------------------------

   --                      Implementation Dependent
   -- These values define the domain limits and are implementation dependent.

   SYSTEM_LIMIT_NUMBER_OF_PARTITIONS     : constant := 32;   -- module scope

   SYSTEM_LIMIT_NUMBER_OF_MESSAGES       : constant := 512;  -- module scope

   SYSTEM_LIMIT_MESSAGE_SIZE             : constant := 8192; -- module scope

   SYSTEM_LIMIT_NUMBER_OF_PROCESSES      : constant := 128;  -- partition scope

   SYSTEM_LIMIT_NUMBER_OF_SAMPLING_PORTS: constant := 512;  -- partition scope

   SYSTEM_LIMIT_NUMBER_OF_QUEUING_PORTS  : constant := 512;  -- partition scope

   SYSTEM_LIMIT_NUMBER_OF_BUFFERS        : constant := 256;  -- partition scope

   SYSTEM_LIMIT_NUMBER_OF_BLACKBOARDS    : constant := 256;  -- partition scope

   SYSTEM_LIMIT_NUMBER_OF_SEMAPHORES     : constant := 256;  -- partition scope

   SYSTEM_LIMIT_NUMBER_OF_EVENTS         : constant := 256;  -- partition scope
```

```
-- ----------------------------
-- Base APEX types           --
-- ----------------------------


--                    Implementation Dependent
-- The actual size of these base types is system specific and the
-- sizes must match the sizes used by the implementation of the
-- underlying Operating System.

type APEX_BYTE is mod 256;
for  APEX_BYTE'size use 8;           -- 8-bit unsigned

type APEX_INTEGER is range -(2**31)..(2**31)-1;
for  APEX_INTEGER'size use 32;       -- 32-bit signed

type APEX_UNSIGNED is mod 2**32;
for  APEX_UNSIGNED'size use 32;      -- 32-bit unsigned

type APEX_LONG_INTEGER is range -(2**63)..(2**63)-1;
for  APEX_LONG_INTEGER'size use 64;  -- 64-bit signed

-- ----------------------------
-- General APEX types        --
-- ----------------------------

type RETURN_CODE_TYPE is (
   NO_ERROR,        -- request valid and operation performed
   NO_ACTION,       -- status of system unaffected by request
   NOT_AVAILABLE,   -- resource required by request unavailable
   INVALID_PARAM,   -- invalid parameter specified in request
   INVALID_CONFIG,  -- parameter incompatible with configuration
   INVALID_MODE,    -- request incompatible with current mode
   TIMED_OUT );     -- time-out tied up with request has expired
for  RETURN_CODE_TYPE use (
   NO_ERROR       => 0,
   NO_ACTION      => 1,
   NOT_AVAILABLE  => 2,
   INVALID_PARAM  => 3,
   INVALID_CONFIG => 4,
   INVALID_MODE   => 5,
   TIMED_OUT      => 6 );
for  RETURN_CODE_TYPE'size use 32;   -- for mapping onto APEX C data size

MAX_NAME_LENGTH      : constant := 30;

type NAME_TYPE       is new STRING (1..MAX_NAME_LENGTH);

type SYSTEM_ADDRESS_TYPE     is new SYSTEM.ADDRESS;

type MESSAGE_ADDR_TYPE       is new SYSTEM.ADDRESS;

type MESSAGE_SIZE_TYPE       is new APEX_INTEGER
                                range 0..SYSTEM_LIMIT_MESSAGE_SIZE;

type MESSAGE_RANGE_TYPE      is new APEX_INTEGER
                                range 0..SYSTEM_LIMIT_NUMBER_OF_MESSAGES;

type PORT_DIRECTION_TYPE     is  (SOURCE, DESTINATION);
```

```
   for  PORT_DIRECTION_TYPE     use (SOURCE => 0, DESTINATION => 1);
   for  PORT_DIRECTION_TYPE'size use 32; -- for mapping onto APEX C data size

   type QUEUING_DISCIPLINE_TYPE is  (FIFO, PRIORITY);
   for  QUEUING_DISCIPLINE_TYPE use (FIFO => 0, PRIORITY => 1);
   for  QUEUING_DISCIPLINE_TYPE'size use 32; -- for mapping onto APEX C
                                             -- data size

   type SYSTEM_TIME_TYPE        is new APEX_TYPES.APEX_LONG_INTEGER;
                                -- 64-bit signed integer with 1 nanoseconds LSB

   INFINITE_TIME_VALUE        : constant SYSTEM_TIME_TYPE := -1;


end APEX_TYPES;
```

```
-- ---------------------------------------------------------------------------
--                                                                         --
-- TIME constant and type definitions and management services              --
--                                                                         --
-- ---------------------------------------------------------------------------

with APEX_TYPES;

package APEX_TIMING is

   procedure TIMED_WAIT (
      DELAY_TIME   : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      RETURN_CODE : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure PERIODIC_WAIT (
      RETURN_CODE : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_TIME (
      SYSTEM_TIME : out APEX_TYPES.SYSTEM_TIME_TYPE;
      RETURN_CODE : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure REPLENISH (
      BUDGET_TIME : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      RETURN_CODE : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_TIMING;

-- ---------------------------------------------------------------------------
--                                                                         --
-- PROCESS constant and type definitions and management services           --
--                                                                         --
-- ---------------------------------------------------------------------------

with APEX_TYPES;

package APEX_PROCESSES is

   MAX_NUMBER_OF_PROCESSES : constant :=
                             APEX_TYPES.SYSTEM_LIMIT_NUMBER_OF_PROCESSES;

   MIN_PRIORITY_VALUE      : constant :=  1;

   MAX_PRIORITY_VALUE      : constant := 239;

   MAX_LOCK_LEVEL          : constant := 16;

   type PROCESS_NAME_TYPE  is new APEX_TYPES.NAME_TYPE;

   type PROCESS_ID_TYPE    is new APEX_TYPES.APEX_INTEGER;
   NULL_PROCESS_ID         : constant PROCESS_ID_TYPE := 0;

   type LOCK_LEVEL_TYPE    is new APEX_TYPES.APEX_INTEGER
                              range 0..MAX_LOCK_LEVEL;

   type STACK_SIZE_TYPE    is new APEX_TYPES.APEX_UNSIGNED;

   type WAITING_RANGE_TYPE is new APEX_TYPES.APEX_INTEGER
                              range 0..MAX_NUMBER_OF_PROCESSES;
```

```
type PRIORITY_TYPE      is new APEX_TYPES.APEX_INTEGER
                           range MIN_PRIORITY_VALUE..MAX_PRIORITY_VALUE;

type PROCESS_STATE_TYPE is  (DORMANT, READY, RUNNING, WAITING);
for  PROCESS_STATE_TYPE use (DORMANT => 0, READY => 1, RUNNING => 2,
                             WAITING => 3);
for  PROCESS_STATE_TYPE'size use 32; -- for mapping on APEX C data size

type DEADLINE_TYPE      is  (SOFT, HARD);
for  DEADLINE_TYPE      use (SOFT => 0, HARD => 1);
for  DEADLINE_TYPE'size use 32;      -- for mapping on APEX C data size

type PROCESS_ATTRIBUTE_TYPE is record
   PERIOD           : APEX_TYPES.SYSTEM_TIME_TYPE;
   TIME_CAPACITY    : APEX_TYPES.SYSTEM_TIME_TYPE;
   ENTRY_POINT      : APEX_TYPES.SYSTEM_ADDRESS_TYPE;
   STACK_SIZE       : STACK_SIZE_TYPE;
   BASE_PRIORITY    : PRIORITY_TYPE;
   DEADLINE         : DEADLINE_TYPE;
   NAME             : PROCESS_NAME_TYPE;
end record;
for  PROCESS_ATTRIBUTE_TYPE use record at mod 8;  -- MAX_NAME_LENGTH dependent
   PERIOD           at  0 range 0..63;
   TIME_CAPACITY    at  8 range 0..63;
    ENTRY_POINT      at 16 range 0..31;           -- alignment on 32-bit word
   STACK_SIZE       at 20 range 0..31;
   BASE_PRIORITY    at 24 range 0..31;
   DEADLINE         at 28 range 0..31;
   NAME             at 32 range 0..30*8-1;        -- 30-char array
end record;

type PROCESS_STATUS_TYPE is record
   DEADLINE_TIME    : APEX_TYPES.SYSTEM_TIME_TYPE;
   CURRENT_PRIORITY : PRIORITY_TYPE;
   PROCESS_STATE    : PROCESS_STATE_TYPE;
   ATTRIBUTES       : PROCESS_ATTRIBUTE_TYPE;
end record;
for  PROCESS_STATUS_TYPE use record at mod 8;     -- MAX_NAME_LENGTH dependent
   DEADLINE_TIME    at  0 range 0..63;            -- 8-byte record
      CURRENT_PRIORITY at  8 range 0..31;
      PROCESS_STATE    at 12 range 0..31;
   ATTRIBUTES       at 16 range 0..62*8-1;        -- 62-byte record
end record;



procedure CREATE_PROCESS (
   ATTRIBUTES     : in  PROCESS_ATTRIBUTE_TYPE;
   PROCESS_ID     : out PROCESS_ID_TYPE;
   RETURN_CODE    : out APEX_TYPES.RETURN_CODE_TYPE );

procedure SET_PRIORITY (
   PROCESS_ID     : in  PROCESS_ID_TYPE;
   PRIORITY       : in  PRIORITY_TYPE;
   RETURN_CODE    : out APEX_TYPES.RETURN_CODE_TYPE );

procedure SUSPEND_SELF (
   TIME_OUT       : in  APEX_TYPES.SYSTEM_TIME_TYPE;
   RETURN_CODE    : out APEX_TYPES.RETURN_CODE_TYPE );
   procedure SUSPEND (
      PROCESS_ID      : in  PROCESS_ID_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure RESUME (
      PROCESS_ID      : in  PROCESS_ID_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure STOP_SELF;

   procedure STOP (
      PROCESS_ID      : in  PROCESS_ID_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure START (
      PROCESS_ID      : in  PROCESS_ID_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure DELAYED_START (
      PROCESS_ID      : in  PROCESS_ID_TYPE;
      DELAY_TIME      : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure LOCK_PREEMPTION (
      LOCK_LEVEL      : out LOCK_LEVEL_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure UNLOCK_PREEMPTION (
      LOCK_LEVEL      : out LOCK_LEVEL_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_MY_ID (
      PROCESS_ID      : out PROCESS_ID_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_PROCESS_ID (
      PROCESS_NAME    : in  PROCESS_NAME_TYPE;
      PROCESS_ID      : out PROCESS_ID_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_PROCESS_STATUS (
      PROCESS_ID      : in  PROCESS_ID_TYPE;
      PROCESS_STATUS  : out PROCESS_STATUS_TYPE;
      RETURN_CODE     : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_PROCESSES;
```

```
-- ---------------------------------------------------------------------------
--                                                                         --
-- PARTITION constant and type definitions and management services        --
--                                                                         --
-- ---------------------------------------------------------------------------

with APEX_TYPES;
with APEX_PROCESSES;

package APEX_PARTITIONS is

   MAX_NUMBER_OF_PARTITIONS : constant :=
                              APEX_TYPES.SYSTEM_LIMIT_NUMBER_OF_PARTITIONS;

   type OPERATING_MODE_TYPE is  (IDLE, COLD_START, WARM_START, NORMAL);
   for  OPERATING_MODE_TYPE use (IDLE       => 0, COLD_START => 1,
                                 WARM_START => 2, NORMAL     => 3);
   for  OPERATING_MODE_TYPE'size use 32; -- for mapping on APEX C data size

   type PARTITION_ID_TYPE   is new APEX_TYPES.APEX_INTEGER;

   type START_CONDITION_TYPE is (
      NORMAL_START,
      PARTITION_RESTART,
      HM_MODULE_RESTART,
      HM_PARTITION_RESTART );
   for  START_CONDITION_TYPE use (
      NORMAL_START         => 0,
      PARTITION_RESTART    => 1,
      HM_MODULE_RESTART    => 2,
      HM_PARTITION_RESTART => 3 );
   for  START_CONDITION_TYPE'size use 32; -- for mapping on APEX C data size

   type PARTITION_STATUS_TYPE is record
      PERIOD          : APEX_TYPES.SYSTEM_TIME_TYPE;
      DURATION        : APEX_TYPES.SYSTEM_TIME_TYPE;
      IDENTIFIER      : PARTITION_ID_TYPE;
      LOCK_LEVEL      : APEX_PROCESSES.LOCK_LEVEL_TYPE;
      OPERATING_MODE  : OPERATING_MODE_TYPE;
      START_CONDITION : START_CONDITION_TYPE;
   end record;
   for  PARTITION_STATUS_TYPE use record at mod 8;
      PERIOD          at  0  range 0..63;
      DURATION        at  8 range 0..63;
      IDENTIFIER      at 16  range 0..31;
      LOCK_LEVEL      at 20 range 0..31;
      OPERATING_MODE  at 24 range 0..31;
      START_CONDITION at 28 range 0..31;
   end record;

   procedure GET_PARTITION_STATUS (
      PARTITION_STATUS : out PARTITION_STATUS_TYPE;
      RETURN_CODE      : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure SET_PARTITION_MODE (
      OPERATING_MODE   : in  OPERATING_MODE_TYPE;
      RETURN_CODE      : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_PARTITIONS;
```

APPENDIX D.1
ADA 83 INTERFACE SPECIFICATION

```
-- -------------------------------------------------------------------------------
--                                                                              --
-- SAMPLING PORT constant and type definitions and management  services        --
--                                                                              --
-- -------------------------------------------------------------------------------

with APEX_TYPES;

package APEX_SAMPLING_PORTS is

   MAX_NUMBER_OF_SAMPLING_PORTS : constant :=
                             APEX_TYPES.SYSTEM_LIMIT_NUMBER_OF_SAMPLING_PORTS;

   type SAMPLING_PORT_NAME_TYPE is new APEX_TYPES.NAME_TYPE;

   type SAMPLING_PORT_ID_TYPE   is new APEX_TYPES.APEX_INTEGER;

   type VALIDITY_TYPE is  (INVALID, VALID);
   for  VALIDITY_TYPE use (INVALID => 0, VALID => 1);
   for  VALIDITY_TYPE'size use 32;     -- for mapping onto APEX C data size

   type SAMPLING_PORT_STATUS_TYPE is record
      REFRESH_PERIOD    : APEX_TYPES.SYSTEM_TIME_TYPE;
      MAX_MESSAGE_SIZE  : APEX_TYPES.MESSAGE_SIZE_TYPE;
      PORT_DIRECTION    : APEX_TYPES.PORT_DIRECTION_TYPE;
      LAST_MSG_VALIDITY : VALIDITY_TYPE;
   end record;
   for  SAMPLING_PORT_STATUS_TYPE use record at mod 8;
      REFRESH_PERIOD    at  0 range 0..63;
      MAX_MESSAGE_SIZE  at  8 range 0..31;
      PORT_DIRECTION    at 12 range 0..31;
      LAST_MSG_VALIDITY at 16 range 0..31;
   end record;



   procedure CREATE_SAMPLING_PORT (
      SAMPLING_PORT_NAME    : in  SAMPLING_PORT_NAME_TYPE;
      MAX_MESSAGE_SIZE      : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      PORT_DIRECTION        : in  APEX_TYPES.PORT_DIRECTION_TYPE;
      REFRESH_PERIOD        : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      SAMPLING_PORT_ID      : out SAMPLING_PORT_ID_TYPE;
      RETURN_CODE           : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure WRITE_SAMPLING_MESSAGE (
      SAMPLING_PORT_ID      : in  SAMPLING_PORT_ID_TYPE;
      MESSAGE_ADDR          : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
      LENGTH                : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      RETURN_CODE           : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure READ_SAMPLING_MESSAGE (
      SAMPLING_PORT_ID      : in  SAMPLING_PORT_ID_TYPE;
      MESSAGE_ADDR          : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
         -- The message address is passed IN, although the respective message
         -- is passed OUT
      LENGTH                : out APEX_TYPES.MESSAGE_SIZE_TYPE;
      VALIDITY              : out VALIDITY_TYPE;
      RETURN_CODE           : out APEX_TYPES.RETURN_CODE_TYPE );
   procedure GET_SAMPLING_PORT_ID (
      SAMPLING_PORT_NAME   : in  SAMPLING_PORT_NAME_TYPE;
      SAMPLING_PORT_ID     : out SAMPLING_PORT_ID_TYPE;
      RETURN_CODE          : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_SAMPLING_PORT_STATUS (
      SAMPLING_PORT_ID     : in  SAMPLING_PORT_ID_TYPE;
      SAMPLING_PORT_STATUS : out SAMPLING_PORT_STATUS_TYPE;
      RETURN_CODE          : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_SAMPLING_PORTS;
```

APPENDIX D.1
ADA 83 INTERFACE SPECIFICATION

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- QUEUING PORT constant and type definitions and management services       --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX_TYPES;
with APEX_PROCESSES;

package APEX_QUEUING_PORTS is

   MAX_NUMBER_OF_QUEUING_PORTS   : constant :=
                              APEX_TYPES.SYSTEM_LIMIT_NUMBER_OF_QUEUING_PORTS;

   type QUEUING_PORT_NAME_TYPE   is new APEX_TYPES.NAME_TYPE;

   type QUEUING_PORT_ID_TYPE     is new APEX_TYPES.APEX_INTEGER;

   type QUEUING_PORT_STATUS_TYPE  is record
      NB_MESSAGE         : APEX_TYPES.MESSAGE_RANGE_TYPE;
      MAX_NB_MESSAGE     : APEX_TYPES.MESSAGE_RANGE_TYPE;
      MAX_MESSAGE_SIZE   : APEX_TYPES.MESSAGE_SIZE_TYPE;
      PORT_DIRECTION     : APEX_TYPES.PORT_DIRECTION_TYPE;
      WAITING_PROCESSES : APEX_PROCESSES.WAITING_RANGE_TYPE;
   end record;
   for  QUEUING_PORT_STATUS_TYPE use record
      NB_MESSAGE         at  0 range 0..31;
      MAX_NB_MESSAGE     at  4 range 0..31;
      MAX_MESSAGE_SIZE   at  8 range 0..31;
      PORT_DIRECTION     at 12 range 0..31;
      WAITING_PROCESSES at 16 range 0..31;
   end record;



   procedure CREATE_QUEUING_PORT (
      QUEUING_PORT_NAME    : in  QUEUING_PORT_NAME_TYPE;
      MAX_MESSAGE_SIZE     : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      MAX_NB_MESSAGE       : in  APEX_TYPES.MESSAGE_RANGE_TYPE;
      PORT_DIRECTION       : in  APEX_TYPES.PORT_DIRECTION_TYPE;
      QUEUING_DISCIPLINE   : in  APEX_TYPES.QUEUING_DISCIPLINE_TYPE;
      QUEUING_PORT_ID      : out QUEUING_PORT_ID_TYPE;
      RETURN_CODE          : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure SEND_QUEUING_MESSAGE (
      QUEUING_PORT_ID      : in  QUEUING_PORT_ID_TYPE;
      MESSAGE_ADDR         : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
      LENGTH               : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      TIME_OUT             : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      RETURN_CODE          : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure RECEIVE_QUEUING_MESSAGE (
      QUEUING_PORT_ID      : in  QUEUING_PORT_ID_TYPE;
      TIME_OUT             : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      MESSAGE_ADDR         : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
         -- The message address is passed IN, although the respective message
         -- is passed OUT
      LENGTH               : out APEX_TYPES.MESSAGE_SIZE_TYPE;
      RETURN_CODE          : out APEX_TYPES.RETURN_CODE_TYPE );
   procedure GET_QUEUING_PORT_ID (
      QUEUING_PORT_NAME    : in  QUEUING_PORT_NAME_TYPE;
      QUEUING_PORT_ID      : out QUEUING_PORT_ID_TYPE;
      RETURN_CODE          : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_QUEUING_PORT_STATUS (
      QUEUING_PORT_ID      : in  QUEUING_PORT_ID_TYPE;
      QUEUING_PORT_STATUS  : out QUEUING_PORT_STATUS_TYPE;
      RETURN_CODE          : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure CLEAR_QUEUING_PORT (
      QUEUING_PORT_ID      : in  QUEUING_PORT_ID_TYPE;
      RETURN_CODE          : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_QUEUING_PORTS;
```

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- BUFFER constant and type definitions and management services            --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX_TYPES;
with APEX_PROCESSES;

package APEX_BUFFERS is

   MAX_NUMBER_OF_BUFFERS   : constant :=
                             APEX_TYPES.SYSTEM_LIMIT_NUMBER_OF_BUFFERS;

   type BUFFER_NAME_TYPE   is new APEX_TYPES.NAME_TYPE;

   type BUFFER_ID_TYPE     is new APEX_TYPES.APEX_INTEGER;

   type BUFFER_STATUS_TYPE is record
      NB_MESSAGE         : APEX_TYPES.MESSAGE_RANGE_TYPE;
      MAX_NB_MESSAGE     : APEX_TYPES.MESSAGE_RANGE_TYPE;
      MAX_MESSAGE_SIZE   : APEX_TYPES.MESSAGE_SIZE_TYPE;
      WAITING_PROCESSES : APEX_PROCESSES.WAITING_RANGE_TYPE;
   end record;
   for  BUFFER_STATUS_TYPE use record
      NB_MESSAGE        at  0 range 0..31;
      MAX_NB_MESSAGE    at  4 range 0..31;
      MAX_MESSAGE_SIZE  at  8 range 0..31;
      WAITING_PROCESSES at 12 range 0..31;
   end record;



   procedure CREATE_BUFFER (
      BUFFER_NAME         : in  BUFFER_NAME_TYPE;
      MAX_MESSAGE_SIZE    : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      MAX_NB_MESSAGE      : in  APEX_TYPES.MESSAGE_RANGE_TYPE;
      QUEUING_DISCIPLINE : in  APEX_TYPES.QUEUING_DISCIPLINE_TYPE;
      BUFFER_ID           : out BUFFER_ID_TYPE;
      RETURN_CODE         : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure SEND_BUFFER (
      BUFFER_ID           : in  BUFFER_ID_TYPE;
      MESSAGE_ADDR        : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
      LENGTH              : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      TIME_OUT            : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      RETURN_CODE         : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure RECEIVE_BUFFER (
      BUFFER_ID           : in  BUFFER_ID_TYPE;
      TIME_OUT            : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      MESSAGE_ADDR        : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
        -- The message address is passed IN, although the respective message
        -- is passed OUT
      LENGTH              : out APEX_TYPES.MESSAGE_SIZE_TYPE;
      RETURN_CODE         : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_BUFFER_ID (
      BUFFER_NAME         : in  BUFFER_NAME_TYPE;
      BUFFER_ID          : out BUFFER_ID_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_BUFFER_STATUS (
      BUFFER_ID          : in  BUFFER_ID_TYPE;
      BUFFER_STATUS      : out BUFFER_STATUS_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_BUFFERS;
```

```
-- -----------------------------------------------------------------------------
--                                                                            --
--   BLACKBOARD constant and type definitions and management services         --
--                                                                            --
-- -----------------------------------------------------------------------------

with APEX_TYPES;
with APEX_PROCESSES;

package APEX_BLACKBOARDS is

   MAX_NUMBER_OF_BLACKBOARDS   : constant :=
                                 APEX_TYPES.SYSTEM_LIMIT_NUMBER_OF_BLACKBOARDS;

   type BLACKBOARD_NAME_TYPE   is new APEX_TYPES.NAME_TYPE;

   type BLACKBOARD_ID_TYPE     is new APEX_TYPES.APEX_INTEGER;

   type EMPTY_INDICATOR_TYPE   is  (EMPTY, OCCUPIED);
   for  EMPTY_INDICATOR_TYPE   use (EMPTY => 0, OCCUPIED => 1);
   for  EMPTY_INDICATOR_TYPE'size use 32; -- for mapping onto APEX C data size

   type BLACKBOARD_STATUS_TYPE is record
      EMPTY_INDICATOR   : EMPTY_INDICATOR_TYPE;
      MAX_MESSAGE_SIZE  : APEX_TYPES.MESSAGE_SIZE_TYPE;
      WAITING_PROCESSES : APEX_PROCESSES.WAITING_RANGE_TYPE;
   end record;
   for  BLACKBOARD_STATUS_TYPE use record
      EMPTY_INDICATOR   at  0 range 0..31;
      MAX_MESSAGE_SIZE  at  4 range 0..31;
      WAITING_PROCESSES at  8 range 0..31;
   end record;



   procedure CREATE_BLACKBOARD (
      BLACKBOARD_NAME   : in  BLACKBOARD_NAME_TYPE;
      MAX_MESSAGE_SIZE  : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      BLACKBOARD_ID     : out BLACKBOARD_ID_TYPE;
      RETURN_CODE       : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure DISPLAY_BLACKBOARD (
      BLACKBOARD_ID     : in  BLACKBOARD_ID_TYPE;
      MESSAGE_ADDR      : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
      LENGTH            : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      RETURN_CODE       : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure READ_BLACKBOARD (
      BLACKBOARD_ID     : in  BLACKBOARD_ID_TYPE;
      TIME_OUT          : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      MESSAGE_ADDR      : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
        -- The message address is passed IN, although the respective message
        -- is passed OUT
      LENGTH            : out APEX_TYPES.MESSAGE_SIZE_TYPE;
      RETURN_CODE       : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure CLEAR_BLACKBOARD (
      BLACKBOARD_ID     : in  BLACKBOARD_ID_TYPE;
      RETURN_CODE       : out APEX_TYPES.RETURN_CODE_TYPE );
   procedure GET_BLACKBOARD_ID (
      BLACKBOARD_NAME   : in  BLACKBOARD_NAME_TYPE;
      BLACKBOARD_ID     : out BLACKBOARD_ID_TYPE;
      RETURN_CODE       : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_BLACKBOARD_STATUS (
      BLACKBOARD_ID     : in  BLACKBOARD_ID_TYPE;
      BLACKBOARD_STATUS : out BLACKBOARD_STATUS_TYPE;
      RETURN_CODE       : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_BLACKBOARDS;
```

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- SEMAPHORE constant and type definitions and management services          --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX_TYPES;
with APEX_PROCESSES;

package APEX_SEMAPHORES is

   MAX_NUMBER_OF_SEMAPHORES   : constant :=
                                APEX_TYPES.SYSTEM_LIMIT_NUMBER_OF_SEMAPHORES;

   MAX_SEMAPHORE_VALUE        : constant := 32767;

   type SEMAPHORE_NAME_TYPE   is new APEX_TYPES.NAME_TYPE;

   type SEMAPHORE_ID_TYPE     is new APEX_TYPES.APEX_INTEGER;

   type SEMAPHORE_VALUE_TYPE  is new APEX_TYPES.APEX_INTEGER
                                 range 0..MAX_SEMAPHORE_VALUE;

   type SEMAPHORE_STATUS_TYPE is record
      CURRENT_VALUE     : SEMAPHORE_VALUE_TYPE;
      MAXIMUM_VALUE     : SEMAPHORE_VALUE_TYPE;
      WAITING_PROCESSES : APEX_PROCESSES.WAITING_RANGE_TYPE;
   end record;
   for  SEMAPHORE_STATUS_TYPE use record
      CURRENT_VALUE     at  0 range 0..31;
      MAXIMUM_VALUE     at  4 range 0..31;
      WAITING_PROCESSES at  8 range 0..31;
   end record;



   procedure CREATE_SEMAPHORE (
      SEMAPHORE_NAME     : in  SEMAPHORE_NAME_TYPE;
      CURRENT_VALUE      : in  SEMAPHORE_VALUE_TYPE;
      MAXIMUM_VALUE      : in  SEMAPHORE_VALUE_TYPE;
      QUEUING_DISCIPLINE : in  APEX_TYPES.QUEUING_DISCIPLINE_TYPE;
      SEMAPHORE_ID       : out SEMAPHORE_ID_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure WAIT_SEMAPHORE (
      SEMAPHORE_ID       : in  SEMAPHORE_ID_TYPE;
      TIME_OUT           : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );
   procedure SIGNAL_SEMAPHORE (
      SEMAPHORE_ID       : in  SEMAPHORE_ID_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_SEMAPHORE_ID (
      SEMAPHORE_NAME     : in  SEMAPHORE_NAME_TYPE;
      SEMAPHORE_ID       : out SEMAPHORE_ID_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_SEMAPHORE_STATUS (
      SEMAPHORE_ID       : in  SEMAPHORE_ID_TYPE;
      SEMAPHORE_STATUS   : out SEMAPHORE_STATUS_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_SEMAPHORES;
```

APPENDIX D.1
ADA 83 INTERFACE SPECIFICATION

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- EVENT constant and type definitions and management services              --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX_TYPES;
with APEX_PROCESSES;

package APEX_EVENTS is

   MAX_NUMBER_OF_EVENTS    : constant :=
                             APEX_TYPES.SYSTEM_LIMIT_NUMBER_OF_EVENTS;

   type EVENT_NAME_TYPE   is new APEX_TYPES.NAME_TYPE;

   type EVENT_ID_TYPE     is new APEX_TYPES.APEX_INTEGER;

   type EVENT_STATE_TYPE  is  (DOWN, UP);
   for  EVENT_STATE_TYPE  use (DOWN => 0, UP => 1);
   for  EVENT_STATE_TYPE'size use 32; -- for mapping on APEX C data size

   type EVENT_STATUS_TYPE is record
      EVENT_STATE       : EVENT_STATE_TYPE;
      WAITING_PROCESSES : APEX_PROCESSES.WAITING_RANGE_TYPE;
   end record;
   for  EVENT_STATUS_TYPE use record
      EVENT_STATE       at  0 range 0..31;
      WAITING_PROCESSES at  4 range 0..31;
   end record;






   procedure CREATE_EVENT (
      EVENT_NAME    : in  EVENT_NAME_TYPE;
      EVENT_ID      : out EVENT_ID_TYPE;
      RETURN_CODE   : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure SET_EVENT (
      EVENT_ID      : in  EVENT_ID_TYPE;
      RETURN_CODE   : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure RESET_EVENT (
      EVENT_ID      : in  EVENT_ID_TYPE;
      RETURN_CODE   : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure WAIT_EVENT (
      EVENT_ID      : in  EVENT_ID_TYPE;
      TIME_OUT      : in  APEX_TYPES.SYSTEM_TIME_TYPE;
      RETURN_CODE   : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_EVENT_ID (
      EVENT_NAME    : in  EVENT_NAME_TYPE;
      EVENT_ID      : out EVENT_ID_TYPE;
      RETURN_CODE   : out APEX_TYPES.RETURN_CODE_TYPE );
   procedure GET_EVENT_STATUS (
      EVENT_ID     : in  EVENT_ID_TYPE;
      EVENT_STATUS : out EVENT_STATUS_TYPE;
      RETURN_CODE  : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_EVENTS;
```

APPENDIX D.1
ADA 83 INTERFACE SPECIFICATION

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- ERROR constant and type definitions and management services             --
--                                                                          --
-- ---------------------------------------------------------------------------

with SYSTEM;
with APEX_TYPES;
with APEX_PROCESSES;

package APEX_HEALTH_MONITORING is

   MAX_ERROR_MESSAGE_SIZE        : constant := 128;

   type ERROR_MESSAGE_SIZE_TYPE  is new APEX_TYPES.APEX_INTEGER
                                    range 0..MAX_ERROR_MESSAGE_SIZE;

   type ERROR_MESSAGE_TYPE       is array (ERROR_MESSAGE_SIZE_TYPE) of
                                    APEX_TYPES.APEX_BYTE;

   type ERROR_CODE_TYPE is (
      DEADLINE_MISSED,
      APPLICATION_ERROR,
      NUMERIC_ERROR,
      ILLEGAL_REQUEST,
      STACK_OVERFLOW,
      MEMORY_VIOLATION,
      HARDWARE_FAULT,
      POWER_FAIL );
   for  ERROR_CODE_TYPE use (
      DEADLINE_MISSED   => 0,
      APPLICATION_ERROR => 1,
      NUMERIC_ERROR     => 2,
      ILLEGAL_REQUEST   => 3,
      STACK_OVERFLOW    => 4,
      MEMORY_VIOLATION  => 5,
      HARDWARE_FAULT    => 6,
      POWER_FAIL        => 7 );
   for  ERROR_CODE_TYPE'size use 32; -- for mapping on APEX C data size

   type ERROR_STATUS_TYPE is record
      ERROR_CODE        : ERROR_CODE_TYPE;
      LENGTH            : ERROR_MESSAGE_SIZE_TYPE;
      FAILED_PROCESS_ID : APEX_PROCESSES.PROCESS_ID_TYPE;
      FAILED_ADDRESS    : APEX_TYPES.SYSTEM_ADDRESS_TYPE;
      MESSAGE           : ERROR_MESSAGE_TYPE;
   end record;
   for  ERROR_STATUS_TYPE use record    -- MAX_ERROR_MESSAGE_SIZE dependent
      ERROR_CODE        at   0 range 0..31;
      LENGTH            at   4 range 0..31;       -- alignment on 32-bit word
      FAILED_PROCESS_ID at   8 range 0..31;
      FAILED_ADDRESS    at  12 range 0..31;
      MESSAGE           at  16 range 0..64*8-1;   -- 64-byte array
   end record;



   procedure REPORT_APPLICATION_MESSAGE (
      MESSAGE_ADDR       : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
      LENGTH             : in  APEX_TYPES.MESSAGE_SIZE_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure CREATE_ERROR_HANDLER (
      ENTRY_POINT        : in  APEX_TYPES.SYSTEM_ADDRESS_TYPE;
      STACK_SIZE         : in  APEX_PROCESSES.STACK_SIZE_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure GET_ERROR_STATUS (
      ERROR_STATUS       : out ERROR_STATUS_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

   procedure RAISE_APPLICATION_ERROR (
      ERROR_CODE         : in  ERROR_CODE_TYPE;
      MESSAGE_ADDR       : in  APEX_TYPES.MESSAGE_ADDR_TYPE;
      LENGTH             : in  ERROR_MESSAGE_SIZE_TYPE;
      RETURN_CODE        : out APEX_TYPES.RETURN_CODE_TYPE );

end APEX_HEALTH_MONITORING;

-- ===============================================================================
```

• Rationale for an Ada 95 binding

Based on code written against the Ada 83 / Ada 95 compatible binding presented in Appendix D.1, a number of improvements have been identified, that lead to cleaner bindings and application code. These improvements take advantage of Ada 95 features.

As in the Ada 83-compatible binding in D.1, this Ada 95 binding preserves representation-compatibility with the C binding in Appendix E. It largely follows the approaches taken in D.1, except as described in the following paragraphs.

There are a few small incompatibilities with Appendix D.1:

- ID types are now private, better reflecting their level of abstraction. If client code relied on the properties of Ada integers, some minor adjustments will be needed.

- There are situations where a context clause for package System may be needed that was unnecessary when various address types were derived from SYSTEM_ADDRESS_TYPE as is done in the Ada 83 binding.

In general, the Ada 95 binding attempts to minimize incompatibilities with the Ada 83 binding in order to limit changes to client code, while providing opportunities for new client code to be written more simply.

A set of wrapper packages are provided for naming compatibility with the Ada 83 binding in D.1.

• Organization

The Ada 95 binding is organized as a hierarchy of library packages rooted in package APEX. The contents of package APEX_TYPES has been moved into the root package, thus reducing the amount of text needed in both the other packages of the binding, and in client code (assuming typical restrictions of the Ada “use” and “use type” clauses).

Further, this organization facilitates addition of child packages of APEX or sub-hierarchies of packages as the Ada standard evolves. Implementation-defined extensions can be added and can be clearly compartmentalized from the portable standard binding while retaining privileged access to the APEX namespace.

Finally, private implementation packages can be provided to support the binding, having privileged access to necessary data, without interfering with the binding standard.

• Types and Subtypes

A number of conventions were established in the Ada 83 binding, based on reasonable assumptions about type usage. In retrospect, some of these decisions turn out to be less than optimal with respect to the writing of client code.

- Use of derived types rather than subtypes.
  There are a number of types derived from foundation APEX types, or from String or Address, that are better defined as subtypes. The important aspect of most of these derived types was their ranges, or their basic characteristics as string or addresses. By defining them as derived types, attributes and operations on types Address and String were lost, as were arithmetic operations on mixed derived integer types. The net result was the need to provide type conversions in situations where no real type security was gained by using derivation rather than subtyping (the types were intended to interoperate, yet no explicit operators had been provided). Such derived types have been replaced with subtypes in the Ada 95 binding.

- ID types.
  In the Ada 83 binding, ID types are represented as derivations of APEX_Integer. The abstraction being modeled is really a private type, as none of the properties of a numeric type is relevant. In the Ada 95 binding, these types are defined as private. Their full definition is as a derivation of APEX_Integer to ensure representation compatibility with the Ada 83 and C bindings. A constant of the type, Null_xxx_ID is provided for initialization of variables, constants, or fields in records. The value of these constants is 0, to correspond with the usual value of Ada “null” and C “NULL”.

- Given representation requirements on Ada 95 compilers, we could have introduced an unconstrained Message_Type with component type APEX_Byte, whose objects could be passed to various routines instead of Message_Address_Type, but there seemed to be little benefit to application code, aside from eliminating some use of ‘Address or ‘Access.

- **Representation**

Ada 95 is significantly stricter and less ambiguous in its representation requirements than Ada 83. The changes are sufficient to eliminate the need for representation clauses for the record types used in the Ada 83 binding, while still guaranteeing the same layout as those given in the Ada 83 and C bindings. We have therefore used the alternative “pragma Convention (C, …)” to reduce clutter in the interfaces while achieving our representation goals. Note that this pragma does not imply an underlying C implementation of the APEX facilities – only that a representation compatible with the target C compiler will be enforced.

The specification also uses the standard Ada 95 packages Interfaces and Interfaces.C to provide APEX types that map directly to the corresponding C predefined types.

It is expected that there may be additional pragmas applied to the entities declared in the specification on a per implementation basis. For example, it is expected that in many implementations pragma Import would be used on most of the subprograms in the binding. Similarly, many implementations may be able to apply pragma Preelaborate to the children of package APEX.

- **Style**

While a relatively small matter, this specification adopts the style of the Ada 95 Reference Manual, as it better reflects modern coding conventions than the Ada 83 style using in the specification of Appendix D.1.

APPENDIX D.2
ADA 95 INTERFACE SPECIFICATION

```
-- This is a compilable Ada 95 specification for the APEX interface,
-- derived from Section 3 of ARINC 653 Part 1.
--
-- The declarations of the services given below are taken from the
-- standard, as are the enumerated types and the names of the others types.
-- Unless specified as implementation dependent, the values specified in
-- this appendix should be implemented as defined.
--
-- All types have defining representation pragmas or clauses to ensure
-- representation compatibility with the C and Ada 83 bindings.

-- ---------------------------------------------------------------------------
--                                                                          --
-- Root package providing constant and type definitions                     --
--                                                                          --
-- ---------------------------------------------------------------------------

with System;
with Interfaces.C;  -- This is the Ada 95 predefined C interface package
package APEX is
   pragma Pure;

   -- ----------------------------
   -- Domain limits             --
   -- ----------------------------

   --                      Implementation Dependent
   --  These values define the domain limits and are implementation dependent.

   System_Limit_Number_Of_Partitions     : constant := 32;
   --  module scope
   System_Limit_Number_Of_Messages       : constant := 512;
   --  module scope
   System_Limit_Message_Size             : constant := 16#10_0000#;
   --  module scope
   System_Limit_Number_Of_Processes      : constant := 1024;
   --  partition scope
   System_Limit_Number_Of_Sampling_Ports : constant := 1024;
   --  partition scope
   System_Limit_Number_Of_Queuing_Ports  : constant := 1024;
   --  partition scope
   System_Limit_Number_Of_Buffers        : constant := 512;
   --  partition scope
   System_Limit_Number_Of_Blackboards    : constant := 512;
   --  partition scope
   System_Limit_Number_Of_Semaphores     : constant := 512;
   --  partition scope
   System_Limit_Number_Of_Events         : constant := 512;
   --  partition scope

   -- ----------------------------
   -- Base APEX types           --
   -- ----------------------------
   --                      Implementation Dependent
   --  The actual sizes of these base types are system-specific and must
   --  match those of the underlying Operating System.

   type APEX_Byte is new Interfaces.C.unsigned_char;     --    8-bit unsigned
   type APEX_Integer is new Interfaces.C.long;           --   32-bit signed
   type APEX_Unsigned is new Interfaces.C.unsigned_long; --   32-bit unsigned

   type APEX_Long_Integer is new Interfaces.Integer_64;  --   64-bit signed
   --  If Integer_64 is not provided in package Interfaces, any implementation-
   --  defined alternative 64-bit signed integer type may be used.

   -- ----------------------------
   -- General APEX types        --
   -- ----------------------------

   type Return_Code_Type is (
      No_Error,        --  request valid and operation performed
      No_Action,       --  status of system unaffected by request
      Not_Available,   --  resource required by request unavailable
      Invalid_Param,   --  invalid parameter specified in request
      Invalid_Config,  --  parameter incompatible with configuration
      Invalid_Mode,    --  request incompatible with current mode
      Timed_Out);      --  time-out tied up with request has expired
   pragma Convention (C, Return_Code_Type);

   Max_Name_Length : constant := 30;

   subtype Name_Type is String (1 .. Max_Name_Length);

   subtype System_Address_Type is System.Address;

   subtype Message_Addr_Type is System.Address;

   subtype Message_Size_Type is APEX_Integer range
      0 .. System_Limit_Message_Size;

   subtype Message_Range_Type is APEX_Integer range
      0 .. System_Limit_Number_Of_Messages;

   type Port_Direction_Type is (Source, Destination);
   pragma Convention (C, Port_Direction_Type);

   type Queuing_Discipline_Type is (Fifo, Priority);
   pragma Convention (C, Queuing_Discipline_Type);

   subtype System_Time_Type is APEX_Long_Integer;
   --  64-bit signed integer with 1 nanosecond LSB

   Infinite_Time_Value : constant System_Time_Type;

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

APPENDIX D.2
ADA 95 INTERFACE SPECIFICATION

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- TIME constant and type definitions and management services               --
--                                                                          --
-- ---------------------------------------------------------------------------

package Apex.Timing is

   procedure Timed_Wait
     (Delay_Time  : in System_Time_Type;
      Return_Code : out Return_Code_Type);

   procedure Periodic_Wait (Return_Code : out Return_Code_Type);

   procedure Get_Time
     (System_Time : out System_Time_Type;
      Return_Code : out Return_Code_Type);

   procedure Replenish
     (Budget_Time : in System_Time_Type;
      Return_Code : out Return_Code_Type);

end Apex.Timing;
```

```
-- ---------------------------------------------------------------------------
--                                                                         --
-- PROCESS constant and type definitions and management services           --
--                                                                         --
-- ---------------------------------------------------------------------------

package APEX.Processes is

   Max_Number_Of_Processes : constant := System_Limit_Number_Of_Processes;
   Min_Priority_Value : constant := 1;
   Max_Priority_Value : constant := 239;
   Max_Lock_Level : constant := 16;

   subtype Process_Name_Type is Name_Type;

   type Process_Id_Type is private;
   Null_Process_Id : constant Process_Id_Type;

   subtype Lock_Level_Type is APEX_Integer range 0 .. Max_Lock_Level;

   subtype Stack_Size_Type is APEX_Unsigned;

   subtype Waiting_Range_Type is APEX_Integer range
     0 .. Max_Number_Of_Processes;

   subtype Priority_Type is APEX_Integer range
     Min_Priority_Value .. Max_Priority_Value;

   type Process_State_Type is (Dormant, Ready, Running, Waiting);

   type Deadline_Type is (Soft, Hard);

   type Process_Attribute_Type is record
      Period         : System_Time_Type;
      Time_Capacity  : System_Time_Type;
      Entry_Point    : System_Address_Type;
      Stack_Size     : Stack_Size_Type;
      Base_Priority  : Priority_Type;
      Deadline       : Deadline_Type;
      Name           : Process_Name_Type;
   end record;

   type Process_Status_Type is record
      Deadline_Time    : System_Time_Type;
      Current_Priority : Priority_Type;
      Process_State    : Process_State_Type;
      Attributes       : Process_Attribute_Type;
   end record;

   procedure Create_Process
     (Attributes  : in Process_Attribute_Type;
      Process_Id  : out Process_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Set_Priority
     (Process_Id  : in Process_Id_Type;
      Priority    : in Priority_Type;
      Return_Code : out Return_Code_Type);
   procedure Suspend_Self
     (Time_Out    : in System_Time_Type;
      Return_Code : out Return_Code_Type);

   procedure Suspend
     (Process_Id  : in Process_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Resume
     (Process_Id  : in Process_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Stop_Self;

   procedure Stop
     (Process_Id  : in Process_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Start
     (Process_Id  : in Process_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Delayed_Start
     (Process_Id  : in Process_Id_Type;
      Delay_Time  : in System_Time_Type;
      Return_Code : out Return_Code_Type);

   procedure Lock_Preemption
     (Lock_Level  : out Lock_Level_Type;
      Return_Code : out Return_Code_Type);

   procedure Unlock_Preemption
     (Lock_Level  : out Lock_Level_Type;
      Return_Code : out Return_Code_Type);

   procedure Get_My_Id
     (Process_Id  : out Process_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Get_Process_Id
     (Process_Name : in Process_Name_Type;
      Process_Id   : out Process_Id_Type;
      Return_Code  : out Return_Code_Type);

   procedure Get_Process_Status
     (Process_Id     : in Process_Id_Type;
      Process_Status : out Process_Status_Type;
      Return_Code    : out Return_Code_Type);

private
   type Process_ID_Type is new APEX_Integer;
   Null_Process_Id : constant Process_Id_Type := 0;

   pragma Convention (C, Process_State_Type);
   pragma Convention (C, Deadline_Type);
   pragma Convention (C, Process_Attribute_Type);
   pragma Convention (C, Process_Status_Type);
end APEX.Processes;
```

```
-- ----------------------------------------------------------------------------
--                                                                          --
-- PARTITION constant and type definitions and management services          --
--                                                                          --
-- ----------------------------------------------------------------------------

with APEX.Processes;
package APEX.Partitions is

   Max_Number_Of_Partitions : constant := System_Limit_Number_Of_Partitions;

   type Operating_Mode_Type is (Idle, Cold_Start, Warm_Start, Normal);

   type Partition_Id_Type is private;
   Null_Partition_Id : constant Partition_Id_Type;

   type Start_Condition_Type is
     (Normal_Start,
      Partition_Restart,
      Hm_Module_Restart,
      Hm_Partition_Restart);

   type Partition_Status_Type is record
      Period          : System_Time_Type;
      Duration        : System_Time_Type;
      Identifier      : Partition_Id_Type;
      Lock_Level      : APEX.Processes.Lock_Level_Type;
      Operating_Mode  : Operating_Mode_Type;
      Start_Condition : Start_Condition_Type;
   end record;

   procedure Get_Partition_Status
     (Partition_Status : out Partition_Status_Type;
      Return_Code      : out Return_Code_Type);

   procedure Set_Partition_Mode
     (Operating_Mode : in Operating_Mode_Type;
      Return_Code    : out Return_Code_Type);

private
   type Partition_ID_Type is new APEX_Integer;
   Null_Partition_Id : constant Partition_Id_Type := 0;

   pragma Convention (C, Operating_Mode_Type);
   pragma Convention (C, Start_Condition_Type);
   pragma Convention (C, Partition_Status_Type);
end APEX.Partitions;
```

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- SAMPLING PORT constant and type definitions and management services      --
--                                                                          --
-- ---------------------------------------------------------------------------

package APEX.Sampling_Ports is

   Max_Number_Of_Sampling_Ports : constant :=
     System_Limit_Number_Of_Sampling_Ports;

   subtype Sampling_Port_Name_Type is Name_Type;

   type Sampling_Port_Id_Type is private;
   Null_Sampling_Port_Id : constant Sampling_Port_Id_Type;

   type Validity_Type is (Invalid, Valid);

   type Sampling_Port_Status_Type is record
      Refresh_Period    : System_Time_Type;
      Max_Message_Size  : Message_Size_Type;
      Port_Direction    : Port_Direction_Type;
      Last_Msg_Validity : Validity_Type;
   end record;

   procedure Create_Sampling_Port
     (Sampling_Port_Name : in Sampling_Port_Name_Type;
      Max_Message_Size   : in Message_Size_Type;
      Port_Direction     : in Port_Direction_Type;
      Refresh_Period     : in System_Time_Type;
      Sampling_Port_Id   : out Sampling_Port_Id_Type;
      Return_Code        : out Return_Code_Type);

   procedure Write_Sampling_Message
     (Sampling_Port_Id : in Sampling_Port_Id_Type;
      Message_Addr     : in Message_Addr_Type;
      Length           : in Message_Size_Type;
      Return_Code      : out Return_Code_Type);

   procedure Read_Sampling_Message
     (Sampling_Port_Id : in Sampling_Port_Id_Type;
      Message_Addr     : in Message_Addr_Type;
   --  The message address is passed IN, although the respective message is
   --  passed OUT
      Length           : out Message_Size_Type;
      Validity         : out Validity_Type;
      Return_Code      : out Return_Code_Type);

   procedure Get_Sampling_Port_Id
     (Sampling_Port_Name : in Sampling_Port_Name_Type;
      Sampling_Port_Id   : out Sampling_Port_Id_Type;
      Return_Code        : out Return_Code_Type);

   procedure Get_Sampling_Port_Status
     (Sampling_Port_Id     : in Sampling_Port_Id_Type;
      Sampling_Port_Status : out Sampling_Port_Status_Type;
      Return_Code          : out Return_Code_Type);
```

```
private
   type Sampling_Port_Id_Type is new APEX_Integer;
   Null_Sampling_Port_Id : constant Sampling_Port_Id_Type := 0;

   pragma Convention (C, Validity_Type);
   pragma Convention (C, Sampling_Port_Status_Type);
end APEX.Sampling_Ports;
```

APPENDIX D.2
ADA 95 INTERFACE SPECIFICATION

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- QUEUING PORT constant and type definitions and management services       --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX.Processes;
package APEX.Queuing_Ports is

   Max_Number_Of_Queuing_Ports : constant :=
     System_Limit_Number_Of_Queuing_Ports;

   subtype Queuing_Port_Name_Type is Name_Type;

   type Queuing_Port_Id_Type is private;
   Null_Queuing_Port_Id : constant Queuing_Port_Id_Type;

   type Queuing_Port_Status_Type is record
      Nb_Message         : Message_Range_Type;
      Max_Nb_Message     : Message_Range_Type;
      Max_Message_Size   : Message_Size_Type;
      Port_Direction     : Port_Direction_Type;
      Waiting_Processes  : APEX.Processes.Waiting_Range_Type;
   end record;

   procedure Create_Queuing_Port
     (Queuing_Port_Name   : in Queuing_Port_Name_Type;
      Max_Message_Size    : in Message_Size_Type;
      Max_Nb_Message      : in Message_Range_Type;
      Port_Direction      : in Port_Direction_Type;
      Queuing_Discipline  : in Queuing_Discipline_Type;
      Queuing_Port_Id     : out Queuing_Port_Id_Type;
      Return_Code         : out Return_Code_Type);

   procedure Send_Queuing_Message
     (Queuing_Port_Id : in Queuing_Port_Id_Type;
      Message_Addr    : in Message_Addr_Type;
      Length          : in Message_Size_Type;
      Time_Out        : in System_Time_Type;
      Return_Code     : out Return_Code_Type);

   procedure Receive_Queuing_Message
     (Queuing_Port_Id : in Queuing_Port_Id_Type;
      Time_Out        : in System_Time_Type;
      Message_Addr    : in Message_Addr_Type;
   --  The message address is passed IN, although the respective message is
   --  passed OUT
      Length          : out Message_Size_Type;
      Return_Code     : out Return_Code_Type);

   procedure Get_Queuing_Port_Id
     (Queuing_Port_Name : in Queuing_Port_Name_Type;
      Queuing_Port_Id   : out Queuing_Port_Id_Type;
      Return_Code       : out Return_Code_Type);

   procedure Get_Queuing_Port_Status
     (Queuing_Port_Id     : in Queuing_Port_Id_Type;
      Queuing_Port_Status : out Queuing_Port_Status_Type;
      Return_Code         : out Return_Code_Type);
   procedure Clear_Queuing_Port
     (Queuing_Port_Id     : in  Queuing_Port_Id_Type;
      Return_Code         : out Return_Code_Type);


private
   type Queuing_Port_Id_Type is new APEX_Integer;
   Null_Queuing_Port_Id : constant Queuing_Port_Id_Type := 0;

   pragma Convention (C, Queuing_Port_Status_Type);
end APEX.Queuing_Ports;
```

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- BUFFER constant and type definitions and management services             --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX.Processes;
package APEX.Buffers is

   Max_Number_Of_Buffers : constant := System_Limit_Number_Of_Buffers;

   subtype Buffer_Name_Type is Name_Type;

   type Buffer_Id_Type is private;
   Null_Buffer_Id : constant Buffer_Id_Type;

   type Buffer_Status_Type is record
      Nb_Message        : Message_Range_Type;
      Max_Nb_Message    : Message_Range_Type;
      Max_Message_Size  : Message_Size_Type;
      Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;

   procedure Create_Buffer
     (Buffer_Name        : in Buffer_Name_Type;
      Max_Message_Size   : in Message_Size_Type;
      Max_Nb_Message     : in Message_Range_Type;
      Queuing_Discipline : in Queuing_Discipline_Type;
      Buffer_Id          : out Buffer_Id_Type;
      Return_Code        : out Return_Code_Type);

   procedure Send_Buffer
     (Buffer_Id    : in Buffer_Id_Type;
      Message_Addr : in Message_Addr_Type;
      Length       : in Message_Size_Type;
      Time_Out     : in System_Time_Type;
      Return_Code  : out Return_Code_Type);

   procedure Receive_Buffer
     (Buffer_Id    : in Buffer_Id_Type;
      Time_Out     : in System_Time_Type;
      Message_Addr : in Message_Addr_Type;
   --  The message address is passed IN, although the respective message is
   --  passed OUT
      Length       : out Message_Size_Type;
      Return_Code  : out Return_Code_Type);

   procedure Get_Buffer_Id
     (Buffer_Name : in Buffer_Name_Type;
      Buffer_Id   : out Buffer_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Get_Buffer_Status
     (Buffer_Id     : in Buffer_Id_Type;
      Buffer_Status : out Buffer_Status_Type;
      Return_Code   : out Return_Code_Type);
```

```
private
   type Buffer_Id_Type is new APEX_Integer;
   Null_Buffer_Id : constant Buffer_Id_Type := 0;

   pragma Convention (C, Buffer_Status_Type);
end APEX.Buffers;
```

APPENDIX D.2
ADA 95 INTERFACE SPECIFICATION

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- BLACKBOARD constant and type definitions and management services         --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX.Processes;
package APEX.Blackboards is

   Max_Number_Of_Blackboards : constant := System_Limit_Number_Of_Blackboards;

   subtype Blackboard_Name_Type is Name_Type;

   type Blackboard_Id_Type is private;
   Null_Blackboard_Id : constant Blackboard_Id_Type;

   type Empty_Indicator_Type is (Empty, Occupied);

   type Blackboard_Status_Type is record
      Empty_Indicator   : Empty_Indicator_Type;
      Max_Message_Size  : Message_Size_Type;
      Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;

   procedure Create_Blackboard
     (Blackboard_Name  : in Blackboard_Name_Type;
      Max_Message_Size : in Message_Size_Type;
      Blackboard_Id    : out Blackboard_Id_Type;
      Return_Code      : out Return_Code_Type);

   procedure Display_Blackboard
     (Blackboard_Id : in Blackboard_Id_Type;
      Message_Addr  : in Message_Addr_Type;
      Length        : in Message_Size_Type;
      Return_Code   : out Return_Code_Type);

   procedure Read_Blackboard
     (Blackboard_Id : in Blackboard_Id_Type;
      Time_Out      : in System_Time_Type;
      Message_Addr  : in Message_Addr_Type;
   --  The message address is passed IN, although the respective message is
   --  passed OUT
      Length        : out Message_Size_Type;
      Return_Code   : out Return_Code_Type);

   procedure Clear_Blackboard
     (Blackboard_Id : in Blackboard_Id_Type;
      Return_Code   : out Return_Code_Type);

   procedure Get_Blackboard_Id
     (Blackboard_Name : in Blackboard_Name_Type;
      Blackboard_Id   : out Blackboard_Id_Type;
      Return_Code     : out Return_Code_Type);

   procedure Get_Blackboard_Status
     (Blackboard_Id     : in Blackboard_Id_Type;
      Blackboard_Status : out Blackboard_Status_Type;
      Return_Code       : out Return_Code_Type);
```

```
private
   type Blackboard_Id_Type is new APEX_Integer;
   Null_Blackboard_Id : constant Blackboard_Id_Type := 0;

   pragma Convention (C, Empty_Indicator_Type);
   pragma Convention (C, Blackboard_Status_Type);
end APEX.Blackboards;
```

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- SEMAPHORE constant and type definitions and management services          --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX.Processes;
package APEX.Semaphores is

   Max_Number_Of_Semaphores : constant := System_Limit_Number_Of_Semaphores;

   Max_Semaphore_Value : constant := 32_767;

   subtype Semaphore_Name_Type is Name_Type;

   type Semaphore_Id_Type is private;
   Null_Semaphore_Id : constant Semaphore_Id_Type;

   type Semaphore_Value_Type is new APEX_Integer range
      0 .. Max_Semaphore_Value;

   type Semaphore_Status_Type is record
      Current_Value     : Semaphore_Value_Type;
      Maximum_Value     : Semaphore_Value_Type;
      Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;

   procedure Create_Semaphore
     (Semaphore_Name     : in Semaphore_Name_Type;
      Current_Value      : in Semaphore_Value_Type;
      Maximum_Value      : in Semaphore_Value_Type;
      Queuing_Discipline : in Queuing_Discipline_Type;
      Semaphore_Id       : out Semaphore_Id_Type;
      Return_Code        : out Return_Code_Type);

   procedure Wait_Semaphore
     (Semaphore_Id : in Semaphore_Id_Type;
      Time_Out     : in System_Time_Type;
      Return_Code  : out Return_Code_Type);

   procedure Signal_Semaphore
     (Semaphore_Id : in Semaphore_Id_Type;
      Return_Code  : out Return_Code_Type);

   procedure Get_Semaphore_Id
     (Semaphore_Name : in Semaphore_Name_Type;
      Semaphore_Id   : out Semaphore_Id_Type;
      Return_Code    : out Return_Code_Type);

   procedure Get_Semaphore_Status
     (Semaphore_Id     : in Semaphore_Id_Type;
      Semaphore_Status : out Semaphore_Status_Type;
      Return_Code      : out Return_Code_Type);

private
   type Semaphore_Id_Type is new APEX_Integer;
   Null_Semaphore_Id : constant Semaphore_Id_Type := 0;

   pragma Convention (C, Semaphore_Status_Type);
```

```
end APEX.Semaphores;
```

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- EVENT constant and type definitions and management services             --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX.Processes;
package APEX.Events is

   Max_Number_Of_Events : constant := System_Limit_Number_Of_Events;

   subtype Event_Name_Type is Name_Type;

   type Event_Id_Type is private;
   Null_Event_Id : constant Event_Id_Type;

   type Event_State_Type is (Down, Up);

   type Event_Status_Type is record
      Event_State       : Event_State_Type;
      Waiting_Processes : APEX.Processes.Waiting_Range_Type;
   end record;

   procedure Create_Event
     (Event_Name  : in Event_Name_Type;
      Event_Id    : out Event_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Set_Event
     (Event_Id    : in Event_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Reset_Event
     (Event_Id    : in Event_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Wait_Event
     (Event_Id    : in Event_Id_Type;
      Time_Out    : in System_Time_Type;
      Return_Code : out Return_Code_Type);

   procedure Get_Event_Id
     (Event_Name  : in Event_Name_Type;
      Event_Id    : out Event_Id_Type;
      Return_Code : out Return_Code_Type);

   procedure Get_Event_Status
     (Event_Id     : in Event_Id_Type;
      Event_Status : out Event_Status_Type;
      Return_Code  : out Return_Code_Type);

private
   type Event_Id_Type is new APEX_Integer;
   Null_Event_Id : constant Event_Id_Type := 0;

   pragma Convention (C, Event_State_Type);
   pragma Convention (C, Event_Status_Type);
end APEX.Events;
```

```
-- ---------------------------------------------------------------------------
--                                                                          --
-- ERROR constant and type definitions and management services              --
--                                                                          --
-- ---------------------------------------------------------------------------

with APEX.Processes;
package APEX.Health_Monitoring is

   Max_Error_Message_Size : constant := 128;

   subtype Error_Message_Size_Type is APEX_Integer range
     1 .. Max_Error_Message_Size;

   type Error_Message_Type is
     array (Error_Message_Size_Type) of APEX_Byte;

   type Error_Code_Type is (
      Deadline_Missed,
      Application_Error,
      Numeric_Error,
      Illegal_Request,
      Stack_Overflow,
      Memory_Violation,
      Hardware_Fault,
      Power_Fail);

   type Error_Status_Type is record
      Error_Code        : Error_Code_Type;
      Length            : Error_Message_Size_Type;
      Failed_Process_Id : APEX.Processes.Process_Id_Type;
      Failed_Address    : System_Address_Type;
      Message           : Error_Message_Type;
   end record;

   procedure Report_Application_Message
     (Message_Addr : in Message_Addr_Type;
      Length       : in Message_Size_Type;
      Return_Code  : out Return_Code_Type);

   procedure Create_Error_Handler
     (Entry_Point : in System_Address_Type;
      Stack_Size  : in APEX.Processes.Stack_Size_Type;
      Return_Code : out Return_Code_Type);

   procedure Get_Error_Status
     (Error_Status : out Error_Status_Type;
      Return_Code  : out Return_Code_Type);

   procedure Raise_Application_Error
     (Error_Code   : in Error_Code_Type;
      Message_Addr : in Message_Addr_Type;
      Length       : in Error_Message_Size_Type;
      Return_Code  : out Return_Code_Type);

private
   pragma Convention (C, Error_Code_Type);
   pragma Convention (C, Error_Status_Type);
end APEX.Health_Monitoring;
```

APPENDIX D.2
ADA 95 INTERFACE SPECIFICATION

```
-- ---------------------------------------------------------------------------
-- --
-- Adaptor packages for compatibility with Ada 83 binding --
-- --
-- ---------------------------------------------------------------------------

with APEX;
package APEX_Types renames APEX;

with APEX.Blackboards;
package APEX_Blackboards renames APEX.Blackboards;

with APEX.Buffers;
package APEX_Buffers renames APEX.Buffers;

with APEX.Events;
package APEX_Events renames APEX.Events;

with APEX.Health_Monitoring;
package APEX_Health_Monitoring renames APEX.Health_Monitoring;

with APEX.Partitions;
package APEX_Partitions renames APEX.Partitions;

with APEX.Processes;
package APEX_Processes renames APEX.Processes;

with APEX.Queuing_Ports;
package APEX_Queuing_Ports renames APEX.Queuing_Ports;

with APEX.Sampling_Ports;
package APEX_Sampling_Ports renames APEX.Sampling_Ports;

with APEX.Semaphores;
package APEX_Semaphores renames APEX.Semaphores;

with APEX.Timing;
package APEX_Timing renames APEX.Timing;
```

APPENDIX E
ANSI C INTERFACE SPECIFICATION

```
/*  This is a compilable ANSI C specification of the APEX interface,        */
/*  derived from Section 3 of ARINC 653 Part 1.                             */

/*  The declarations of the services given below are taken from the         */
/*  standard, as are the enum types and the names of the others types.      */
/*  Unless specified as implementation dependent, the values specified in   */
/*  this appendix should be implemented as defined.                         */

/*  This ANSI C specification follows the same structure (package) as      */
/*  the Ada specification. The objective is to have the same definitions    */
/*  and representations of the interface in both the Ada and C languages.   */
/*  The two specifications are aligned in the same order, and define the    */
/*  same items.                                                             */

/*----------------------------------------------------------------*/
/*                                                                */
/* Global constant and type definitions                           */
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_TYPES
#define APEX_TYPES

/*---------------------------*/
/* Domain limits             */
/*---------------------------*/

/*                       Implementation Dependent                           */
/* These values define the domain limits and are implementation dependent. */

#define  SYSTEM_LIMIT_NUMBER_OF_PARTITIONS      32      /* module scope */

#define  SYSTEM_LIMIT_NUMBER_OF_MESSAGES        512     /* module scope */

#define  SYSTEM_LIMIT_MESSAGE_SIZE              8192    /* module scope */

#define  SYSTEM_LIMIT_NUMBER_OF_PROCESSES       128     /* partition scope */

#define  SYSTEM_LIMIT_NUMBER_OF_SAMPLING_PORTS  512     /* partition scope */

#define  SYSTEM_LIMIT_NUMBER_OF_QUEUING_PORTS   512     /* partition scope */

#define  SYSTEM_LIMIT_NUMBER_OF_BUFFERS         256     /* partition scope */

#define  SYSTEM_LIMIT_NUMBER_OF_BLACKBOARDS     256     /* partition scope */

#define  SYSTEM_LIMIT_NUMBER_OF_SEMAPHORES      256     /* partition scope */

#define  SYSTEM_LIMIT_NUMBER_OF_EVENTS          256     /* partition scope */
```

```
/*----------------------*/
/* Base APEX types      */
/*----------------------*/

/*                       Implementation Dependent                      */
/*  The actual size of these base types is system specific and the     */
/*  sizes must match the sizes used by the implementation of the       */
/*  underlying Operating System.                                       */

typedef  unsigned char   APEX_BYTE;             /* 8-bit unsigned  */

typedef  long            APEX_INTEGER;          /* 32-bit signed   */

typedef  unsigned long   APEX_UNSIGNED;         /* 32-bit unsigned */

typedef  long long       APEX_LONG_INTEGER;     /* 64-bit signed   */

/*----------------------*/
/* General APEX types   */
/*----------------------*/

typedef
   enum {
        NO_ERROR       = 0,   /* request valid and operation performed      */
        NO_ACTION      = 1,   /* status of system unaffected by request     */
        NOT_AVAILABLE  = 2,   /* resource required by request unavailable   */
        INVALID_PARAM  = 3,   /* invalid parameter specified in request     */
        INVALID_CONFIG = 4,   /* parameter incompatible with configuration  */
        INVALID_MODE   = 5,   /* request incompatible with current mode     */
        TIMED_OUT      = 6    /* time-out tied up with request has expired  */
   } RETURN_CODE_TYPE;

#define  MAX_NAME_LENGTH          30

typedef  char            NAME_TYPE[MAX_NAME_LENGTH];

typedef  void            (* SYSTEM_ADDRESS_TYPE);

typedef  APEX_BYTE *     MESSAGE_ADDR_TYPE;

typedef  APEX_INTEGER    MESSAGE_SIZE_TYPE;

typedef  APEX_INTEGER    MESSAGE_RANGE_TYPE;

typedef  enum { SOURCE = 0, DESTINATION = 1 } PORT_DIRECTION_TYPE;

typedef  enum { FIFO = 0, PRIORITY = 1 } QUEUING_DISCIPLINE_TYPE;

typedef  APEX_LONG_INTEGER   SYSTEM_TIME_TYPE;
                           /* 64-bit signed integer with a 1 nanosecond LSB */

#define  INFINITE_TIME_VALUE    -1

#endif
```

```
/*----------------------------------------------------------------*/
/*                                                                */
/* TIME constant and type definitions and management services     */
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_TIME
#define APEX_TIME

extern void TIMED_WAIT (
       /*in */ SYSTEM_TIME_TYPE         DELAY_TIME,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void PERIODIC_WAIT (
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_TIME (
       /*out*/ SYSTEM_TIME_TYPE         *SYSTEM_TIME,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void REPLENISH (
       /*in */ SYSTEM_TIME_TYPE         BUDGET_TIME,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

#endif

/*----------------------------------------------------------------*/
/*                                                                */
/* PROCESS constant and type definitions and management services  */
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_PROCESS
#define APEX_PROCESS

#define  MAX_NUMBER_OF_PROCESSES  SYSTEM_LIMIT_NUMBER_OF_PROCESSES

#define  MIN_PRIORITY_VALUE       1

#define  MAX_PRIORITY_VALUE       239

#define  MAX_LOCK_LEVEL           16

typedef  NAME_TYPE            PROCESS_NAME_TYPE;

typedef  APEX_INTEGER         PROCESS_ID_TYPE;
#define  NULL_PROCESS_ID          0

typedef  APEX_INTEGER         LOCK_LEVEL_TYPE;

typedef  APEX_UNSIGNED        STACK_SIZE_TYPE;

typedef  APEX_INTEGER         WAITING_RANGE_TYPE;

typedef  APEX_INTEGER         PRIORITY_TYPE;
```

```
typedef
   enum {
        DORMANT  = 0,
        READY    = 1,
        RUNNING  = 2,
        WAITING  = 3
   } PROCESS_STATE_TYPE;

typedef  enum { SOFT = 0, HARD = 1 } DEADLINE_TYPE;

typedef
   struct {
      SYSTEM_TIME_TYPE      PERIOD;
      SYSTEM_TIME_TYPE      TIME_CAPACITY;
      SYSTEM_ADDRESS_TYPE   ENTRY_POINT;
      STACK_SIZE_TYPE       STACK_SIZE;
      PRIORITY_TYPE         BASE_PRIORITY;
      DEADLINE_TYPE         DEADLINE;
      PROCESS_NAME_TYPE     NAME;
   } PROCESS_ATTRIBUTE_TYPE;

typedef
   struct {
      SYSTEM_TIME_TYPE        DEADLINE_TIME;
      PRIORITY_TYPE           CURRENT_PRIORITY;
      PROCESS_STATE_TYPE      PROCESS_STATE;
      PROCESS_ATTRIBUTE_TYPE  ATTRIBUTES;
   } PROCESS_STATUS_TYPE;


extern void CREATE_PROCESS (
       /*in */ PROCESS_ATTRIBUTE_TYPE   *ATTRIBUTES,
       /*out*/ PROCESS_ID_TYPE          *PROCESS_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void SET_PRIORITY (
       /*in */ PROCESS_ID_TYPE          PROCESS_ID,
       /*in */ PRIORITY_TYPE            PRIORITY,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void SUSPEND_SELF (
       /*in */ SYSTEM_TIME_TYPE         TIME_OUT,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void SUSPEND (
       /*in */ PROCESS_ID_TYPE          PROCESS_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void RESUME (
       /*in */ PROCESS_ID_TYPE          PROCESS_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void STOP_SELF (void);

extern void STOP (
       /*in */ PROCESS_ID_TYPE          PROCESS_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );
```

```
extern void START (
       /*in */ PROCESS_ID_TYPE          PROCESS_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void DELAYED_START (
       /*in */ PROCESS_ID_TYPE          PROCESS_ID,
       /*in */ SYSTEM_TIME_TYPE         DELAY_TIME,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void LOCK_PREEMPTION (
       /*out*/ LOCK_LEVEL_TYPE          *LOCK_LEVEL,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void UNLOCK_PREEMPTION (
       /*out*/ LOCK_LEVEL_TYPE          *LOCK_LEVEL,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_MY_ID (
       /*out*/ PROCESS_ID_TYPE          *PROCESS_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_PROCESS_ID (
       /*in */ PROCESS_NAME_TYPE        PROCESS_NAME,
       /*out*/ PROCESS_ID_TYPE          *PROCESS_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_PROCESS_STATUS (
       /*in */ PROCESS_ID_TYPE          PROCESS_ID,
       /*out*/ PROCESS_STATUS_TYPE      *PROCESS_STATUS,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

#endif
```

```
/*----------------------------------------------------------------*/
/*                                                                */
/* PARTITION constant and type definitions and management services*/
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_PARTITION
#define APEX_PARTITION

#define  MAX_NUMBER_OF_PARTITIONS  SYSTEM_LIMIT_NUMBER_OF_PARTITIONS

typedef
   enum {
        IDLE       = 0,
        COLD_START = 1,
        WARM_START = 2,
        NORMAL     = 3
   } OPERATING_MODE_TYPE;

typedef  APEX_INTEGER    PARTITION_ID_TYPE;

typedef
   enum {
        NORMAL_START         = 0,
        PARTITION_RESTART    = 1,
        HM_MODULE_RESTART    = 2,
        HM_PARTITION_RESTART = 3
   } START_CONDITION_TYPE;

typedef
   struct {
      SYSTEM_TIME_TYPE     PERIOD;
      SYSTEM_TIME_TYPE     DURATION;
      PARTITION_ID_TYPE    IDENTIFIER;
      LOCK_LEVEL_TYPE      LOCK_LEVEL;
      OPERATING_MODE_TYPE  OPERATING_MODE;
      START_CONDITION_TYPE START_CONDITION;
   } PARTITION_STATUS_TYPE;


extern void GET_PARTITION_STATUS (
       /*out*/ PARTITION_STATUS_TYPE    *PARTITION_STATUS,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void SET_PARTITION_MODE (
       /*in */ OPERATING_MODE_TYPE      OPERATING_MODE,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

#endif
```

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* SAMPLING PORT constant and type definitions and management services*/
/*                                                                   */
/*-------------------------------------------------------------------*/

#ifndef APEX_SAMPLING
#define APEX_SAMPLING

#define  MAX_NUMBER_OF_SAMPLING_PORTS   SYSTEM_LIMIT_NUMBER_OF_SAMPLING_PORTS

typedef  NAME_TYPE       SAMPLING_PORT_NAME_TYPE;

typedef  APEX_INTEGER    SAMPLING_PORT_ID_TYPE;

typedef  enum { INVALID = 0, VALID = 1 } VALIDITY_TYPE;

typedef
   struct {
      SYSTEM_TIME_TYPE         REFRESH_PERIOD;
      MESSAGE_SIZE_TYPE        MAX_MESSAGE_SIZE;
      PORT_DIRECTION_TYPE      PORT_DIRECTION;
      VALIDITY_TYPE            LAST_MSG_VALIDITY;
   } SAMPLING_PORT_STATUS_TYPE;



extern void CREATE_SAMPLING_PORT (
       /*in */ SAMPLING_PORT_NAME_TYPE    SAMPLING_PORT_NAME,
       /*in */ MESSAGE_SIZE_TYPE          MAX_MESSAGE_SIZE,
       /*in */ PORT_DIRECTION_TYPE        PORT_DIRECTION,
       /*in */ SYSTEM_TIME_TYPE           REFRESH_PERIOD,
       /*out*/ SAMPLING_PORT_ID_TYPE      *SAMPLING_PORT_ID,
       /*out*/ RETURN_CODE_TYPE           *RETURN_CODE );

extern void WRITE_SAMPLING_MESSAGE (
       /*in */ SAMPLING_PORT_ID_TYPE      SAMPLING_PORT_ID,
       /*in */ MESSAGE_ADDR_TYPE          MESSAGE_ADDR,     /* by reference */
       /*in */ MESSAGE_SIZE_TYPE          LENGTH,
       /*out*/ RETURN_CODE_TYPE           *RETURN_CODE );

extern void READ_SAMPLING_MESSAGE (
       /*in */ SAMPLING_PORT_ID_TYPE      SAMPLING_PORT_ID,
       /*out*/ MESSAGE_ADDR_TYPE          MESSAGE_ADDR,
       /*out*/ MESSAGE_SIZE_TYPE          *LENGTH,
       /*out*/ VALIDITY_TYPE              *VALIDITY,
       /*out*/ RETURN_CODE_TYPE           *RETURN_CODE );

extern void GET_SAMPLING_PORT_ID (
       /*in */ SAMPLING_PORT_NAME_TYPE    SAMPLING_PORT_NAME,
       /*out*/ SAMPLING_PORT_ID_TYPE      *SAMPLING_PORT_ID,
       /*out*/ RETURN_CODE_TYPE           *RETURN_CODE );

extern void GET_SAMPLING_PORT_STATUS (
       /*in */ SAMPLING_PORT_ID_TYPE      SAMPLING_PORT_ID,
       /*out*/ SAMPLING_PORT_STATUS_TYPE  *SAMPLING_PORT_STATUS,
       /*out*/ RETURN_CODE_TYPE           *RETURN_CODE );

#endif
```

```
/*---------------------------------------------------------------------*/
/*                                                                     */
/* QUEUING PORT constant and type definitions and management services */
/*                                                                     */
/*---------------------------------------------------------------------*/

#ifndef APEX_QUEUING
#define APEX_QUEUING

#define  MAX_NUMBER_OF_QUEUING_PORTS    SYSTEM_LIMIT_NUMBER_OF_QUEUING_PORTS

typedef  NAME_TYPE       QUEUING_PORT_NAME_TYPE;

typedef  APEX_INTEGER    QUEUING_PORT_ID_TYPE;

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;
   } QUEUING_PORT_STATUS_TYPE;


extern void CREATE_QUEUING_PORT (
       /*in */ QUEUING_PORT_NAME_TYPE   QUEUING_PORT_NAME,
       /*in */ MESSAGE_SIZE_TYPE        MAX_MESSAGE_SIZE,
       /*in */ MESSAGE_RANGE_TYPE       MAX_NB_MESSAGE,
       /*in */ PORT_DIRECTION_TYPE      PORT_DIRECTION,
       /*in */ QUEUING_DISCIPLINE_TYPE  QUEUING_DISCIPLINE,
       /*out*/ QUEUING_PORT_ID_TYPE     *QUEUING_PORT_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void SEND_QUEUING_MESSAGE (
       /*in */ QUEUING_PORT_ID_TYPE     QUEUING_PORT_ID,
       /*in */ MESSAGE_ADDR_TYPE        MESSAGE_ADDR,       /* by reference */
       /*in */ MESSAGE_SIZE_TYPE        LENGTH,
       /*in */ SYSTEM_TIME_TYPE         TIME_OUT,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE);

extern void RECEIVE_QUEUING_MESSAGE (
       /*in */ QUEUING_PORT_ID_TYPE     QUEUING_PORT_ID,
       /*in */ SYSTEM_TIME_TYPE         TIME_OUT,
       /*out*/ MESSAGE_ADDR_TYPE        MESSAGE_ADDR,
       /*out*/ MESSAGE_SIZE_TYPE        *LENGTH,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_QUEUING_PORT_ID (
       /*in */ QUEUING_PORT_NAME_TYPE   QUEUING_PORT_NAME,
       /*out*/ QUEUING_PORT_ID_TYPE     *QUEUING_PORT_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_QUEUING_PORT_STATUS (
       /*in */ QUEUING_PORT_ID_TYPE     QUEUING_PORT_ID,
       /*out*/ QUEUING_PORT_STATUS_TYPE *QUEUING_PORT_STATUS,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );
```

```
extern void CLEAR_QUEUING_PORT (
       /*in */ QUEUING_PORT_ID_TYPE     QUEUING_PORT_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

#endif
```

```
/*----------------------------------------------------------------*/
/*                                                                */
/* BUFFER constant and type definitions and management services   */
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_BUFFER
#define APEX_BUFFER

#define  MAX_NUMBER_OF_BUFFERS    SYSTEM_LIMIT_NUMBER_OF_BUFFERS

typedef  NAME_TYPE       BUFFER_NAME_TYPE;

typedef  APEX_INTEGER    BUFFER_ID_TYPE;

typedef
   struct {
      MESSAGE_RANGE_TYPE  NB_MESSAGE;
      MESSAGE_RANGE_TYPE  MAX_NB_MESSAGE;
      MESSAGE_SIZE_TYPE   MAX_MESSAGE_SIZE;
      WAITING_RANGE_TYPE  WAITING_PROCESSES;
   } BUFFER_STATUS_TYPE;



extern void CREATE_BUFFER (
       /*in */ BUFFER_NAME_TYPE         BUFFER_NAME,
       /*in */ MESSAGE_SIZE_TYPE        MAX_MESSAGE_SIZE,
       /*in */ MESSAGE_RANGE_TYPE       MAX_NB_MESSAGE,
       /*in */ QUEUING_DISCIPLINE_TYPE  QUEUING_DISCIPLINE,
       /*out*/ BUFFER_ID_TYPE           *BUFFER_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void SEND_BUFFER (
       /*in */ BUFFER_ID_TYPE           BUFFER_ID,
       /*in */ MESSAGE_ADDR_TYPE        MESSAGE_ADDR,       /* by reference */
       /*in */ MESSAGE_SIZE_TYPE        LENGTH,
       /*in */ SYSTEM_TIME_TYPE         TIME_OUT,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void RECEIVE_BUFFER (
       /*in */ BUFFER_ID_TYPE           BUFFER_ID,
       /*in */ SYSTEM_TIME_TYPE         TIME_OUT,
       /*out*/ MESSAGE_ADDR_TYPE        MESSAGE_ADDR,
       /*out*/ MESSAGE_SIZE_TYPE        *LENGTH,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_BUFFER_ID (
       /*in */ BUFFER_NAME_TYPE         BUFFER_NAME,
       /*out*/ BUFFER_ID_TYPE           *BUFFER_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_BUFFER_STATUS (
       /*in */ BUFFER_ID_TYPE           BUFFER_ID,
       /*out*/ BUFFER_STATUS_TYPE       *BUFFER_STATUS,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

#endif
```

```
/*-------------------------------------------------------------------*/
/*                                                                   */
/* BLACKBOARD constant and type definitions and management services  */
/*                                                                   */
/*-------------------------------------------------------------------*/

#ifndef APEX_BLACKBOARD
#define APEX_BLACKBOARD

#define  MAX_NUMBER_OF_BLACKBOARDS      SYSTEM_LIMIT_NUMBER_OF_BLACKBOARDS

typedef  NAME_TYPE       BLACKBOARD_NAME_TYPE;

typedef  APEX_INTEGER    BLACKBOARD_ID_TYPE;

typedef  enum { EMPTY = 0, OCCUPIED = 1 } EMPTY_INDICATOR_TYPE;

typedef
   struct {
      EMPTY_INDICATOR_TYPE  EMPTY_INDICATOR;
      MESSAGE_SIZE_TYPE     MAX_MESSAGE_SIZE;
      WAITING_RANGE_TYPE    WAITING_PROCESSES;
   } BLACKBOARD_STATUS_TYPE;


extern void CREATE_BLACKBOARD (
       /*in */ BLACKBOARD_NAME_TYPE      BLACKBOARD_NAME,
       /*in */ MESSAGE_SIZE_TYPE         MAX_MESSAGE_SIZE,
       /*out*/ BLACKBOARD_ID_TYPE        *BLACKBOARD_ID,
       /*out*/ RETURN_CODE_TYPE          *RETURN_CODE );

extern void DISPLAY_BLACKBOARD (
       /*in */ BLACKBOARD_ID_TYPE        BLACKBOARD_ID,
       /*in */ MESSAGE_ADDR_TYPE         MESSAGE_ADDR,       /* by reference */
       /*in */ MESSAGE_SIZE_TYPE         LENGTH,
       /*out*/ RETURN_CODE_TYPE          *RETURN_CODE );

extern void READ_BLACKBOARD (
       /*in */ BLACKBOARD_ID_TYPE        BLACKBOARD_ID,
       /*in */ SYSTEM_TIME_TYPE          TIME_OUT,
       /*out*/ MESSAGE_ADDR_TYPE         MESSAGE_ADDR,
       /*out*/ MESSAGE_SIZE_TYPE         *LENGTH,
       /*out*/ RETURN_CODE_TYPE          *RETURN_CODE );

extern void CLEAR_BLACKBOARD (
       /*in */ BLACKBOARD_ID_TYPE        BLACKBOARD_ID,
       /*out*/ RETURN_CODE_TYPE          *RETURN_CODE );

extern void GET_BLACKBOARD_ID (
       /*in */ BLACKBOARD_NAME_TYPE      BLACKBOARD_NAME,
       /*out*/ BLACKBOARD_ID_TYPE        *BLACKBOARD_ID,
       /*out*/ RETURN_CODE_TYPE          *RETURN_CODE );

extern void GET_BLACKBOARD_STATUS (
       /*in */ BLACKBOARD_ID_TYPE        BLACKBOARD_ID,
       /*out*/ BLACKBOARD_STATUS_TYPE    *BLACKBOARD_STATUS,
       /*out*/ RETURN_CODE_TYPE          *RETURN_CODE );

#endif
```

```
/*----------------------------------------------------------------*/
/*                                                                */
/* SEMAPHORE constant and type definitions and management services*/
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_SEMAPHORE
#define APEX_SEMAPHORE

#define  MAX_NUMBER_OF_SEMAPHORES  SYSTEM_LIMIT_NUMBER_OF_SEMAPHORES

#define  MAX_SEMAPHORE_VALUE       32767

typedef  NAME_TYPE       SEMAPHORE_NAME_TYPE;

typedef  APEX_INTEGER    SEMAPHORE_ID_TYPE;

typedef  APEX_INTEGER    SEMAPHORE_VALUE_TYPE;

typedef
   struct {
      SEMAPHORE_VALUE_TYPE  CURRENT_VALUE;
      SEMAPHORE_VALUE_TYPE  MAXIMUM_VALUE;
      WAITING_RANGE_TYPE    WAITING_PROCESSES;
   } SEMAPHORE_STATUS_TYPE;



extern void CREATE_SEMAPHORE (
       /*in */ SEMAPHORE_NAME_TYPE      SEMAPHORE_NAME,
       /*in */ SEMAPHORE_VALUE_TYPE     CURRENT_VALUE,
       /*in */ SEMAPHORE_VALUE_TYPE     MAXIMUM_VALUE,
       /*in */ QUEUING_DISCIPLINE_TYPE  QUEUING_DISCIPLINE,
       /*out*/ SEMAPHORE_ID_TYPE        *SEMAPHORE_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void WAIT_SEMAPHORE (
       /*in */ SEMAPHORE_ID_TYPE        SEMAPHORE_ID,
       /*in */ SYSTEM_TIME_TYPE         TIME_OUT,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void SIGNAL_SEMAPHORE (
       /*in */ SEMAPHORE_ID_TYPE        SEMAPHORE_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_SEMAPHORE_ID (
       /*in */ SEMAPHORE_NAME_TYPE      SEMAPHORE_NAME,
       /*out*/ SEMAPHORE_ID_TYPE        *SEMAPHORE_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_SEMAPHORE_STATUS (
       /*in */ SEMAPHORE_ID_TYPE        SEMAPHORE_ID,
       /*out*/ SEMAPHORE_STATUS_TYPE    *SEMAPHORE_STATUS,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

#endif
```

```
/*----------------------------------------------------------------*/
/*                                                                */
/* EVENT constant and type definitions and management services    */
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_EVENT
#define APEX_EVENT

#define  MAX_NUMBER_OF_EVENTS     SYSTEM_LIMIT_NUMBER_OF_EVENTS

typedef  NAME_TYPE       EVENT_NAME_TYPE;

typedef  APEX_INTEGER    EVENT_ID_TYPE;

typedef  enum { DOWN = 0, UP = 1 } EVENT_STATE_TYPE;

typedef
   struct {
      EVENT_STATE_TYPE    EVENT_STATE;
      WAITING_RANGE_TYPE  WAITING_PROCESSES;
   } EVENT_STATUS_TYPE;



extern void CREATE_EVENT (
       /*in */ EVENT_NAME_TYPE          EVENT_NAME,
       /*out*/ EVENT_ID_TYPE            *EVENT_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void SET_EVENT (
       /*in */ EVENT_ID_TYPE            EVENT_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void RESET_EVENT (
       /*in */ EVENT_ID_TYPE            EVENT_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void WAIT_EVENT (
       /*in */ EVENT_ID_TYPE            EVENT_ID,
       /*in */ SYSTEM_TIME_TYPE         TIME_OUT,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_EVENT_ID (
       /*in */ EVENT_NAME_TYPE          EVENT_NAME,
       /*out*/ EVENT_ID_TYPE            *EVENT_ID,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_EVENT_STATUS (
       /*in */ EVENT_ID_TYPE            EVENT_ID,
       /*out*/ EVENT_STATUS_TYPE        *EVENT_STATUS,
       /*out*/ RETURN_CODE_TYPE         *RETURN_CODE );

#endif
```

```
/*----------------------------------------------------------------*/
/*                                                                */
/* ERROR constant and type definitions and management services    */
/*                                                                */
/*----------------------------------------------------------------*/

#ifndef APEX_ERROR
#define APEX_ERROR

#define  MAX_ERROR_MESSAGE_SIZE         128

typedef  APEX_INTEGER   ERROR_MESSAGE_SIZE_TYPE;

typedef  APEX_BYTE      ERROR_MESSAGE_TYPE[MAX_ERROR_MESSAGE_SIZE];

typedef
   enum {
        DEADLINE_MISSED   = 0,
        APPLICATION_ERROR = 1,
        NUMERIC_ERROR     = 2,
        ILLEGAL_REQUEST   = 3,
        STACK_OVERFLOW    = 4,
        MEMORY_VIOLATION  = 5,
        HARDWARE_FAULT    = 6,
        POWER_FAIL        = 7
   } ERROR_CODE_TYPE;

typedef
   struct {
      ERROR_CODE_TYPE         ERROR_CODE;
      ERROR_MESSAGE_SIZE_TYPE LENGTH;
      PROCESS_ID_TYPE         FAILED_PROCESS_ID;
      SYSTEM_ADDRESS_TYPE     FAILED_ADDRESS;
      ERROR_MESSAGE_TYPE      MESSAGE;
   } ERROR_STATUS_TYPE;


extern void REPORT_APPLICATION_MESSAGE (
       /*in */   MESSAGE_ADDR_TYPE        MESSAGE_ADDR,
       /*in */   MESSAGE_SIZE_TYPE        LENGTH,
       /*out*/   RETURN_CODE_TYPE         *RETURN_CODE );

extern void CREATE_ERROR_HANDLER (
       /*in */   SYSTEM_ADDRESS_TYPE      ENTRY_POINT,
       /*in */   STACK_SIZE_TYPE          STACK_SIZE,
       /*out*/   RETURN_CODE_TYPE         *RETURN_CODE );

extern void GET_ERROR_STATUS (
       /*out*/   ERROR_STATUS_TYPE        *ERROR_STATUS,
       /*out*/   RETURN_CODE_TYPE         *RETURN_CODE );

extern void RAISE_APPLICATION_ERROR (
       /*in */   ERROR_CODE_TYPE          ERROR_CODE,
       /*in */   MESSAGE_ADDR_TYPE        MESSAGE_ADDR,
       /*in */   ERROR_MESSAGE_SIZE_TYPE  LENGTH,
       /*out*/   RETURN_CODE_TYPE         *RETURN_CODE );

#endif
/*==============================================================================*/
```

APPENDIX F
APEX SERVICE RETURN CODE MATRIX

(This Appendix deleted by Supplement 1.)

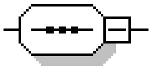
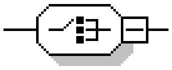
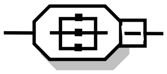
APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA TYPES

This appendix presents a graphical representation of the ARINC 653 XML-schema types defined by this standard. A tool was used to generate graphics shown here. There are many commercial-off-the-shelf XML tools which provide a graphical interface for building XML-Schema **and the XML instance file.** For example**, XMLSpy 2009** was used in this appendix to generate graphics.

Figures 1 through **10** graphically depict the ARINC 653 XML-Schema types. The figures include annotations on the meaning of each tagged element.

Table 1 below defines the meanings of the graphical symbols shown in the Figures 1 through **10**. The symbols are an artifact of the tool XMLSpy and are not part of the XML Standard.

Table 1 – Graphical Symbols

	Corresponds to the XML definition ‘xsd:sequence’ element. All child elements must appear in sequential order.
	Corresponds to the XML definition ‘xsd:choice’ element. Only one of the child elements can be chosen.
	Corresponds to the XML definition ‘xsd:all’ element. All the child elements can appear in any order.
tag name	Corresponds to an XML Tag also called an element. Used to tag the data in the instance file.
type name	Corresponds to a created type definition. Used to build up the schema and to reuse common definitions.
any	Corresponds to the XML definition ‘xsd:any’. Allows user defined extensions to the schema.

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA TYPES

The following type definitions, see Table 2, 3 and 4, are used throughout the schema. Other type definitions that are used to define multiple elements or attributes are shown at their first occurrence.

The NameType is defined as shown in Table 2.

Table 2 – NameType Definition

Simple Type	Name Type
Type	restriction of xs:string
Facets	minLength 1 maxLength 30

The DecOrHexValueType allows numbers to be entered as integer **decimal** or hex values. Hex values are preceded by “0x”.

Table 3 – DecOrHexValueType Definition

Simple Type	DecOrHexValueType
Type	restriction of xs:string
Facets	pattern **[+-]{0,1}[0-9]+|0x[0-9a-fA-F]+**

The IdentifierValueType allows **identifiers to be entered as integer or hex values. Hex values are preceded by “0x”.**

Table – 4 IdentifierValueType Definition

Simple Type	IdentifierValueType
Type	DexOrHexValueType
Facets	pattern **[+-]{0,1}[0-9]+|0x[0-9a-fA-F]+**

complexType A653_ErrorIdentifierType

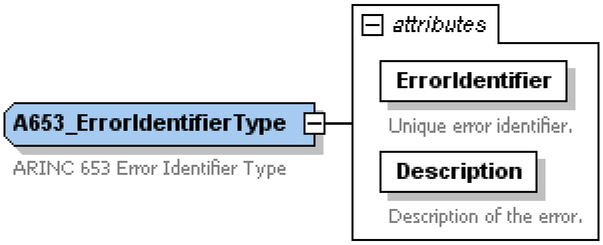
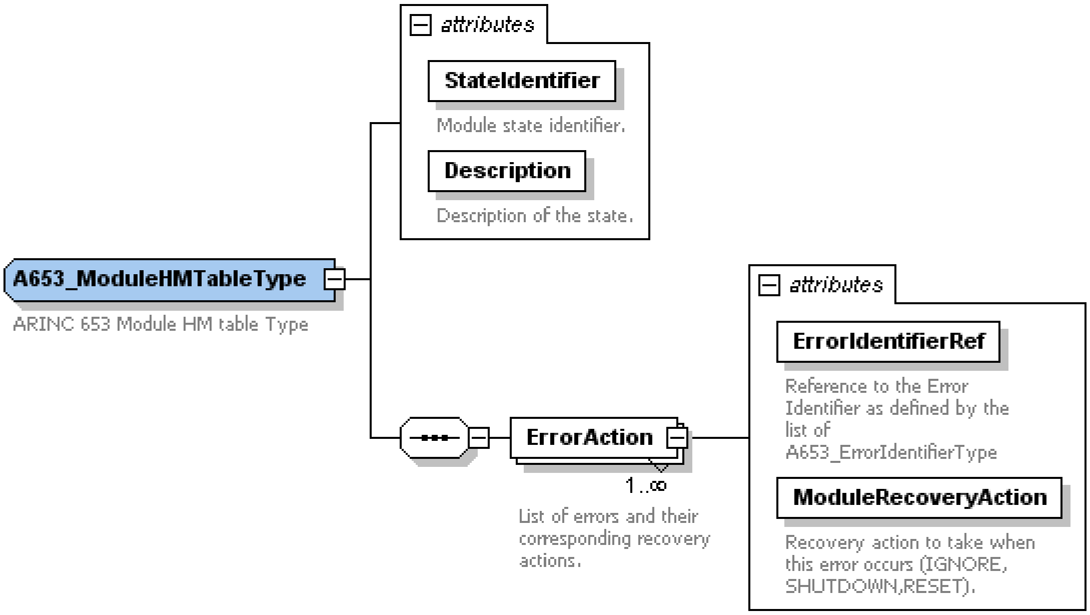
Diagram	 A653_ErrorIdentifierType ARINC 653 Error Identifier Type attributes ErrorIdentifier Unique error identifier. Description Description of the error.				
Attributes	Name	Type	Use	Definition	Scope
	ErrorIdentifier	IdentifierValueType	Required	Unique error identifier.	The module
	Description	String	Required	Description of the error.	The error

Figure 1 – A653_ErrorIdentifierType Definitions

complexType A653_ModuleHMTableType

Diagram	 attributes StateIdentifier Module state identifier. Description Description of the state. A653_ModuleHMTableType ARINC 653 Module HM table Type ErrorAction 1..∞ List of errors and their corresponding recovery actions. attributes ErrorIdentifierRef Reference to the Error Identifier as defined by the list of A653_ErrorIdentifierType ModuleRecoveryAction Recovery action to take when this error occurs (IGNORE, SHUTDOWN,RESET).				
Attributes	**Name**	**Type**	**Use**	**Definition**	**Scope**
	StateIdentifier	**IdentifierValueType**	**Required**	**The module state identifier.**	**The module**
	Description	**String**	**Required**	**A text description of the state.**	**The module**
	ErrorIdentifierRef	**IdentifierValueType**	**Required**	**Reference to the identification of the error. Each systems error is described by A653_ErrorIdentifierType. Constraints: Only the errors managed by the core software are described.**	**The module**
	ModuleRecoveryAction	**ModuleLevelError RecoveryAction Type**	**Required**	**The Module recovery action to apply for that error and system state.**	**The module**

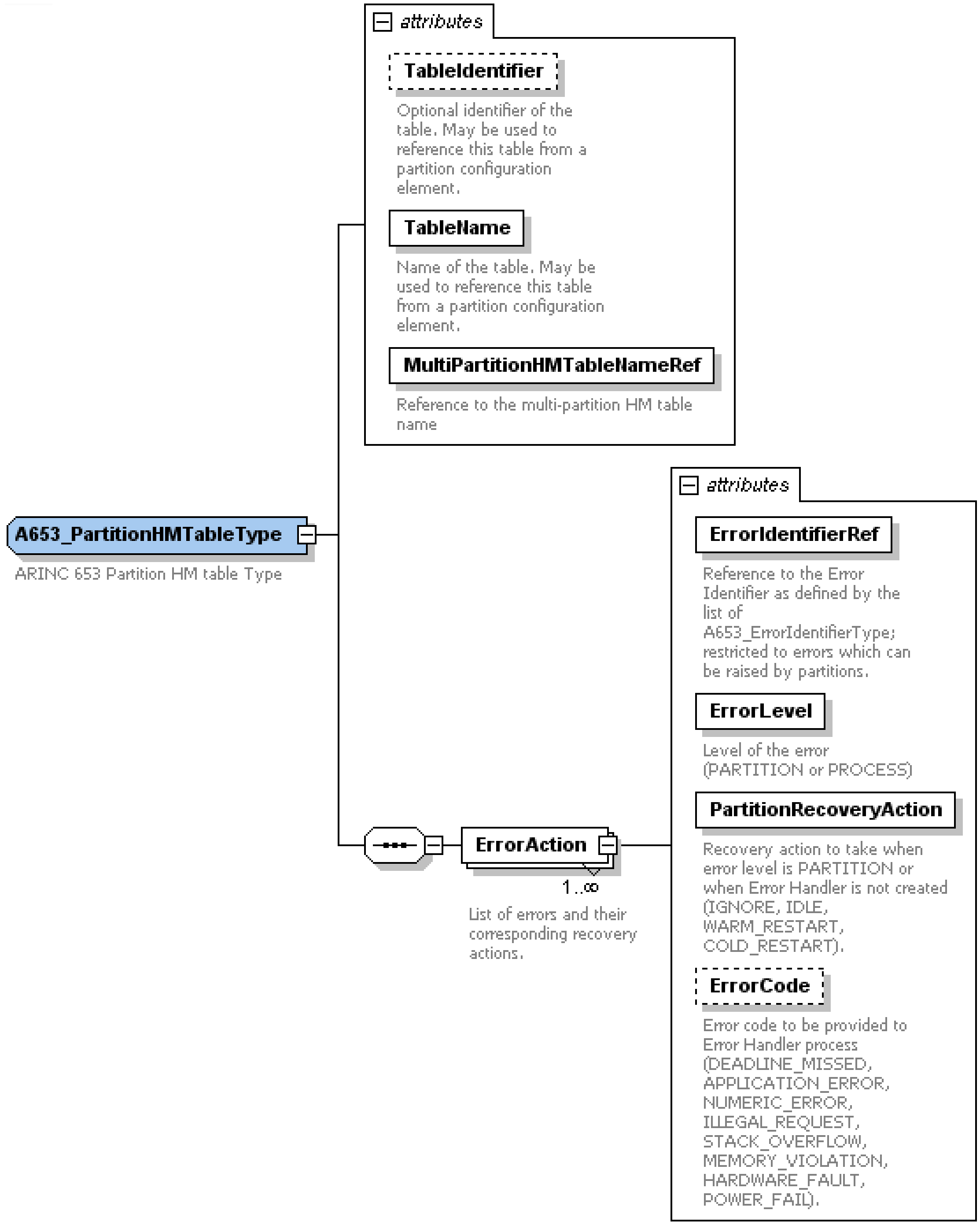
simpleType ModuleLevelErrorRecoveryActionType

Type	**Restriction of xs:string**		
Facets	enumeration	**IGNORE**	**Constraint: This type could be extended to allow specific recovery action, but the following recovery actions should be kept.**
	enumeration	**SHUTDOWN**	
	enumeration	**RESET**	

Figure 2 – A653_ModuleHMTableType & ModuleLevelErrorRecoveryActionType Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA TYPES

complexType A653_PartitionHMTableType

Diagram	 attributes TableIdentifier Optional identifier of the table. May be used to reference this table from a partition configuration element. TableName Name of the table. May be used to reference this table from a partition configuration element. MultiPartitionHMTableNameRef Reference to the multi-partition HM table name A653_PartitionHMTableType ARINC 653 Partition HM table Type ErrorAction 1..∞ List of errors and their corresponding recovery actions. attributes ErrorIdentifierRef Reference to the Error Identifier as defined by the list of A653_ErrorIdentifierType; restricted to errors which can be raised by partitions. ErrorLevel Level of the error (PARTITION or PROCESS) PartitionRecoveryAction Recovery action to take when error level is PARTITION or when Error Handler is not created (IGNORE, IDLE, WARM_RESTART, COLD_RESTART). ErrorCode Error code to be provided to Error Handler process (DEADLINE_MISSED, APPLICATION_ERROR, NUMERIC_ERROR, ILLEGAL_REQUEST, STACK_OVERFLOW, MEMORY_VIOLATION, HARDWARE_FAULT, POWER_FAIL).

Attributes	Name	Type	Use	Definition	Scope
	TableIdentifier	IdentifierValueType	Optional	The identification of the table in the system. Constraint: This identifier must be unique in the core module.	The module
	TableName	NameType	Required	The name of the table. Constraint: This name must be unique in the core module.	The module
	MultiPartitionHMTableHameRef	NameType	Required	Reference to name of the multi-partition HM table.	The Partition HM Table
	ErrorIdentifierRef	IdentifierValueType	Required	Reference to the identification of the error. Each systems error is described by A653_ErrorIdentifierType. Constraint: Only the errors allocated to the partitions are described.	The Partition HM Table
	ErrorLevel	PartitionHMTable ErrorLevelType	Required	Level of the recovery action that should be applied: PARTITION or PROCESS.	The Partition HM Table
	Partition RecoveryAction	PartitionLevelError RecoveryAction Type	Required	The recovery action or the default recovery action to apply.	The Partition HM Table
	ErrorCode	ErrorCodeType	Optional	Error Code that will be given to the error process to identify this error.	The Partition HM Table

Figure 3 – A653_PartitionHMTableType Definition

simpleType PartitionLevelErrorRecoveryActionType

Type	Restriction of xs:string	
Facets	enumeration	IGNORE
	enumeration	IDLE
	enumeration	WARM_RESTART
	enumeration	COLD_RESTART

simpleType PartitionHMTableErrorLevelType

Type	Restriction of xs:string	
Facets	enumeration	PARTITION
	enumeration	PROCESS

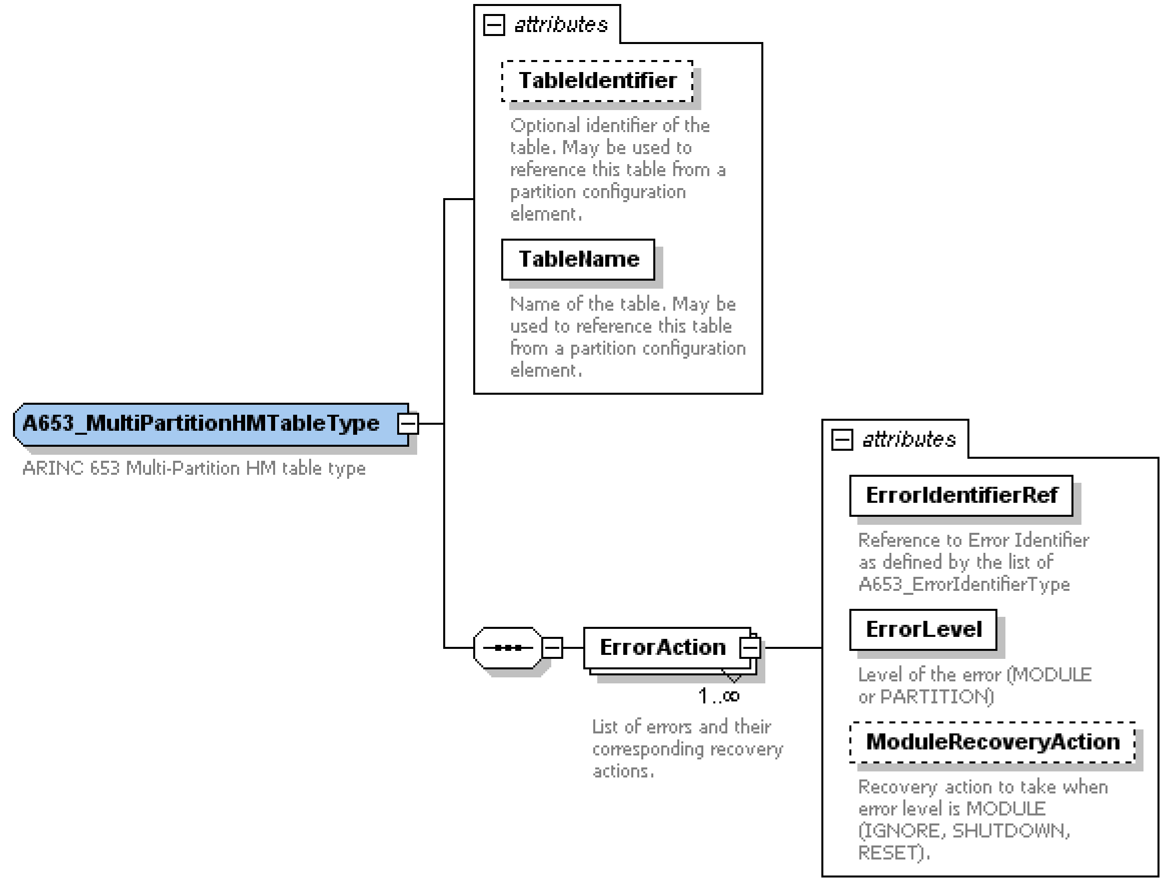
simpleType ErrorCodeType

Type	Restriction of xs:string	
Facets	enumeration	DEADLINE_MISSED
	enumeration	APPLICATION_ERROR
	enumeration	NUMERIC_ERROR
	enumeration	ILLEGAL_REQUEST
	enumeration	STACK_OVERFLOW
	enumeration	MEMORY_VIOLATION
	enumeration	HARDWARE_FAULT
	enumeration	POWER_FAIL

Figure 4 – PartitionLevelErrorRecoveryActionType, PartitionHMTableErrorLevelType & ErrorCodeType Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA TYPES

complexType A653_MultiPartitionHMTableType

Diagram	 attributes TableIdentifier Optional identifier of the table. May be used to reference this table from a partition configuration element. TableName Name of the table. May be used to reference this table from a partition configuration element. A653_MultiPartitionHMTableType ARINC 653 Multi-Partition HM table type ErrorAction 1..∞ List of errors and their corresponding recovery actions. attributes ErrorIdentifierRef Reference to Error Identifier as defined by the list of A653_ErrorIdentifierType ErrorLevel Level of the error (MODULE or PARTITION) ModuleRecoveryAction Recovery action to take when error level is MODULE (IGNORE, SHUTDOWN, RESET).

Attributes	Name	Type	Use	Definition	Scope
	TableIdentifier	IdentifierValueType	Optional	The identification of the table in the system. Constraint: This Identifier must be unique in the core module.	The module
	TableName	NameTypes	Required	The name of the table. Constraint: This name must be unique in the core module.	The module
	ErrorIdentifierRef	IdentifierValueType	Required	Reference to the identification of the error. Each systems error is described by A653_ErrorIdentifierType. Constraint: All the errors that could be raised inside the partition windows have to be described.	The multi-partition HM table
	ErrorLevel	MultiPartitionHMTable ErrorLevelType	Required	Level of the recovery action that should be applied: PARTITION or MODULE.	The multi-partition HM table
	ModuleRecovery Action	ModuleLevelError RecoveryActionType	Optional	Recovery action to apply in case of a MODULE recovery action.	The multi-partition HM table

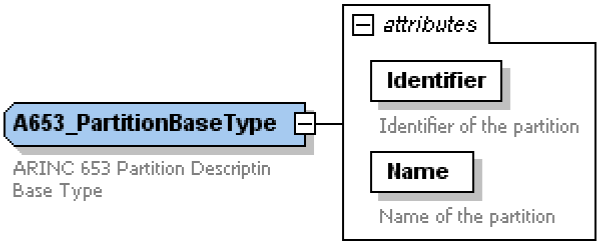
simpleType MultiPartitionHMTableErrorLevelType

Type	Restriction of xs:string	
Facets	enumeration	MODULE
	enumeration	PARTITION

Figure 5 – A653_MultiPartitionHMTableType & MultiPartitionHMTableErrorLevelType Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA TYPES

complexType A653_PartitionBaseType

Diagram	 attributes A653_PartitionBaseType ARINC 653 Partition Descriptin Base Type Identifier Identifier of the partition Name Name of the partition				
Attributes	**Name**	**Type**	**Use**	**Definition**	**Scope**
	Identifier	**IdentifierValueType**	Required	**The partition identification. Constraint: The Identifier should be unique through the entire module.**	**The module**
	Name	NameType	**Required**	**The partition name. Constraint: The name should be unique through the entire module.**	**The module**

complexType A653_PartitionPeriodicityType

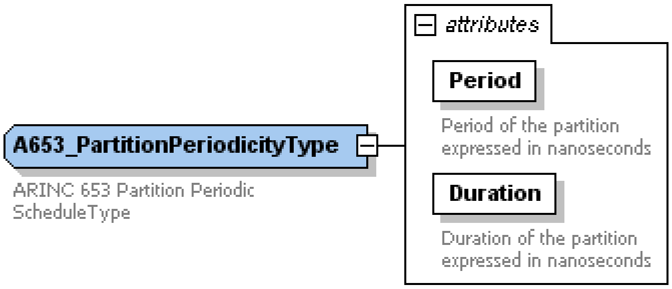
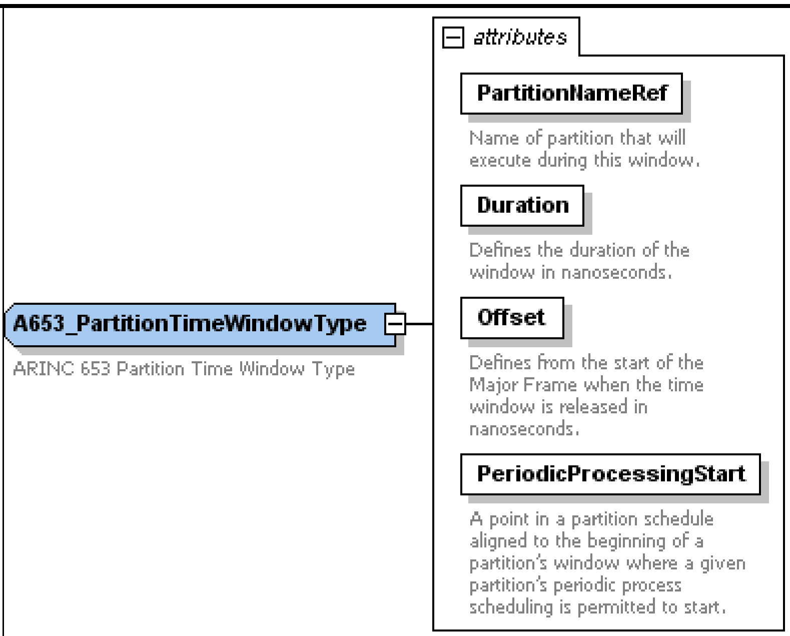
Diagram	 attributes A653_PartitionPeriodicityType ARINC 653 Partition Periodic ScheduleType Period Period of the partition expressed in nanoseconds Duration Duration of the partition expressed in nanoseconds				
Attributes	**Name**	**Type**	**Use**	**Definition**	**Scope**
	Period	**DecOrHexValueType**	**Required**	**The partition period expressed in nanoseconds.**	**The partition**
	Duration	**DecOrHexValueType**	**Required**	**The partition duration expressed in nanoseconds.**	**The partition**

Figure 6 – A653_PartitionBaseType and A653_PartitionPeriodicityType Definitions

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA TYPES

complexType A653_PartitionTimeWindowType

Diagram

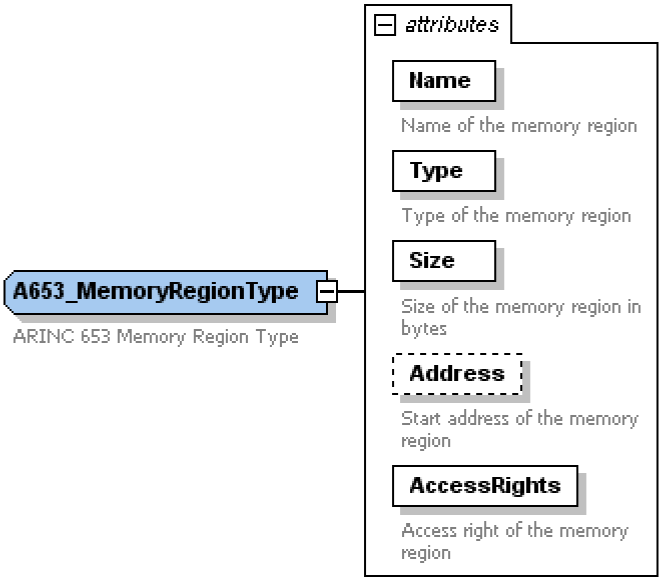

attributes
PartitionNameRef
Name of partition that will execute during this window.
Duration
Defines the duration of the window in nanoseconds.
Offset
Defines from the start of the Major Frame when the time window is released in nanoseconds.
PeriodicProcessingStart
A point in a partition schedule aligned to the beginning of a partition's window where a given partition's periodic process scheduling is permitted to start.
A653_PartitionTimeWindowType
ARINC 653 Partition Time Window Type

Attributes	Name	Type	Use	Definition	Scope
	PartitionNameRef	NameType	Required	The name of the partition executed during this time window.	The module
	Duration	DecOrHexValueType	Required	The duration of the time window expressed in nanoseconds.	The partition time window
	Offset	DecOrHexValueType	Required	The offset from the start of the Major Frame when the time window starts expressed in nanoseconds.	The partition time window
	PeriodicProcessing Start	Boolean	Required	A point in a partition schedule aligned to the beginning of a partition's window where a given partition's periodic process scheduling is permitted to start.	The partition time Window

Figure 7 – A653_PartitionTimeWindowType Definition

APPENDIX G
GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA TYPES

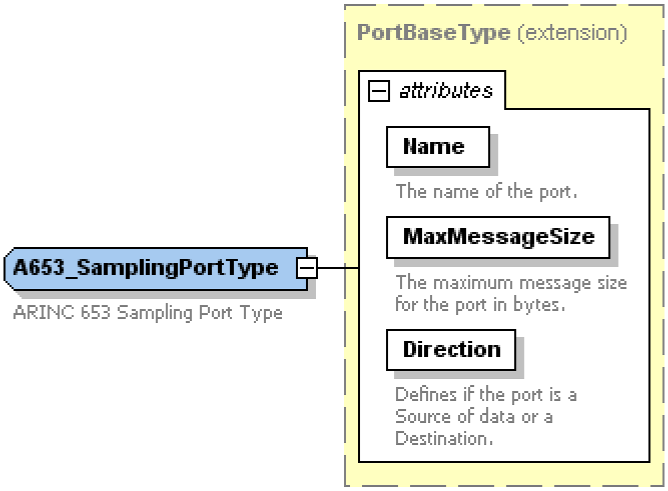
complexType A653_MemoryRegionType

Diagram					
Attributes	Name	Type	Use	Definition	Scope
	Name	NameType	Required	The memory region name.	The partition
	Type	String	Required	The memory region type.	The memory region
	Size	DecOrHexValueType	Required	The memory region size.	The memory region
	Address	DecOrHexValueType	Optional	The memory region address.	The memory region
	AccessRights	String	Required	The partition access rights to the memory region.	The memory region

A653_MemoryRegionType
ARINC 653 Memory Region Type
attributes
Name
Name of the memory region
Type
Type of the memory region
Size
Size of the memory region in bytes
Address
Start address of the memory region
AccessRights
Access right of the memory region

Figure 8 – A653_MemoryRegionType Definition

complexType A653_SamplingPortType

Diagram	 PortBaseType (extension) attributes Name The name of the port. MaxMessageSize The maximum message size for the port in bytes. Direction Defines if the port is a Source of data or a Destination. A653_SamplingPortType ARINC 653 Sampling Port Type				
Attributes	**Name**	**Type**	**Use**	**Definition**	**Scope**
	Name	NameType	Required	The name of the port	The sampling port
	MaxMessageSize	DecOrHexValueType	Required	The maximum number of bytes a message may contain.	The sampling port
	Direction	PortDirectionType	Required	The direction of the port	The sampling port

simpleType PortDirectionType

Type	**Restriction of xs:string**
Facets	enumeration SOURCE enumeration DESTINATION

Figure 9 – A653_SamplingPortType & PortDirectionType Definitions

complexType A653_QueuingPortType

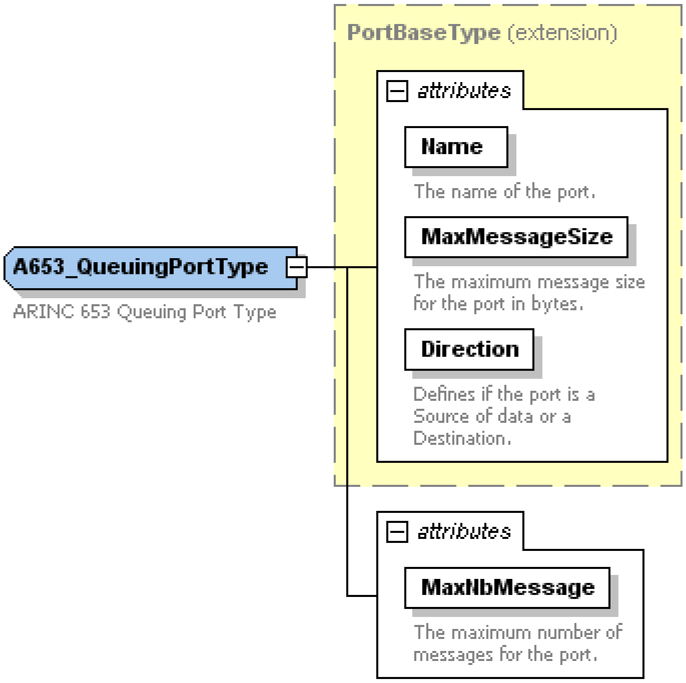
Diagram	 PortBaseType (extension) attributes Name The name of the port. MaxMessageSize The maximum message size for the port in bytes. Direction Defines if the port is a Source of data or a Destination. A653_QueuingPortType ARINC 653 Queuing Port Type attributes MaxNbMessage The maximum number of messages for the port.				
Attributes	Name	Type	Use	Definition	Scope
	Name	NameType	Required	The name of the port.	The queuing port
	MaxMessageSize	DecOrHexValueType	Required	The maximum number of bytes a message may contain.	The queuing port
	Direction	PortDirectionType	Required	The direction of the port.	The queuing port
	MaxNbMessage	DecOrHexValueType	Required	The maximum number of messages in the port.	The queuing port

Figure 10 – A653_QueuingPortType Definition

The ARINC 653 XML-Schema type code is divided into 2 files: ARINC653Types.xsd and ARINC653TypesExtensible.xsd.

ARINC653Types.xsd contains most of the ARINC653 XML types and should not altered by the OS implementer.

ARINC653TypeExtensible.xsd contains types that can be extended and is included by ARINC653Types.xsd

ARINC653Types.xsd :

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:schema xmlns="ARINC653" xmlns:ar="ARINC653" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="ARINC653" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation="ARINC653TypesExtensible.xsd"/>
    <xs:simpleType name="NameType">
        <xs:annotation>
            <xs:documentation>Name Type</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="30"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="DecOrHexValueType">
        <xs:annotation>
            <xs:documentation>Decimal or Hexadecimal Type</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:pattern value="[+-]{0,1}[0-9]+|0x[0-9a-fA-F]+"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="IdentifierValueType">
        <xs:annotation>
            <xs:documentation>Identifier Value Type</xs:documentation>
        </xs:annotation>
        <xs:restriction base="DecOrHexValueType"/>
    </xs:simpleType>
    <xs:simpleType name="PortDirectionType">
        <xs:annotation>
            <xs:documentation>Port Direction Enumeration Type</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="SOURCE"/>
            <xs:enumeration value="DESTINATION"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ErrorCodeType">
        <xs:annotation>
            <xs:documentation>Predefined ARINC 653 process level error codes.</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="DEADLINE_MISSED"/>
            <xs:enumeration value="APPLICATION_ERROR"/>
            <xs:enumeration value="NUMERIC_ERROR"/>
            <xs:enumeration value="ILLEGAL_REQUEST"/>
            <xs:enumeration value="STACK_OVERFLOW"/>
            <xs:enumeration value="MEMORY_VIOLATION"/>
            <xs:enumeration value="HARDWARE_FAULT"/>
            <xs:enumeration value="POWER_FAIL"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="PartitionLevelErrorRecoveryActionType">
```

```
        <xs:annotation>
            <xs:documentation>Error recovery actions to take when partition level errors occur
Type</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="IGNORE"/>
            <xs:enumeration value="IDLE"/>
            <xs:enumeration value="WARM_RESTART"/>
            <xs:enumeration value="COLD_RESTART"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="MultiPartitionHMTableErrorLevelType">
        <xs:annotation>
            <xs:documentation>Error levels for Multi-Partition HM table (MODULE or
PARTITION).</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="MODULE"/>
            <xs:enumeration value="PARTITION"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="PartitionHMTableErrorLevelType">
        <xs:annotation>
            <xs:documentation>Error levels for Partition HM table (PARTITION or PROCESS).</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="PARTITION"/>
            <xs:enumeration value="PROCESS"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="PortBaseType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Base Port Type</xs:documentation>
        </xs:annotation>
        <xs:attribute name="Name" type="NameType" use="required">
            <xs:annotation>
                <xs:documentation>The name of the port.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="MaxMessageSize" type="DecOrHexValueType" use="required">
            <xs:annotation>
                <xs:documentation>The maximum message size for the port in bytes.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Direction" type="PortDirectionType" use="required">
            <xs:annotation>
                <xs:documentation>Defines if the port is a Source of data or a
Destination.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="A653_QueuingPortType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Queuing Port Type </xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="PortBaseType">
                <xs:attribute name="MaxNbMessage" type="DecOrHexValueType" use="required">
                    <xs:annotation>
                        <xs:documentation>The maximum number of messages for the
port.</xs:documentation>
                    </xs:annotation>
                </xs:attribute>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="A653_SamplingPortType">
```

```
        <xs:annotation>
            <xs:documentation>ARINC 653 Sampling Port Type</xs:documentation>
        </xs:annotation>
        <xs:complexContent>
            <xs:extension base="PortBaseType"/>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="A653_PartitionBaseType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Partition Description Base Type</xs:documentation>
        </xs:annotation>
        <xs:attribute name="Identifier" type="IdentifierValueType" use="required">
            <xs:annotation>
                <xs:documentation>Identifier of the partition</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Name" type="NameType" use="required">
            <xs:annotation>
                <xs:documentation>Name of the partition</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="A653_PartitionPeriodicityType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Partition Periodic ScheduleType</xs:documentation>
        </xs:annotation>
        <xs:attribute name="Period" type="DecOrHexValueType" use="required">
            <xs:annotation>
                <xs:documentation>Period of the partition expressed in nanoseconds</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Duration" type="DecOrHexValueType" use="required">
            <xs:annotation>
                <xs:documentation>Duration of the partition expressed in nanoseconds</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="A653_PartitionTimeWindowType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Partition Time Window Type</xs:documentation>
        </xs:annotation>
        <xs:attribute name="PartitionNameRef" type="NameType" use="required">
            <xs:annotation>
                <xs:documentation>Name of partition that will execute during this window.
</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Duration" type="DecOrHexValueType" use="required">
            <xs:annotation>
                <xs:documentation>Defines the duration of the window in nanoseconds.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Offset" type="DecOrHexValueType" use="required">
            <xs:annotation>
                <xs:documentation>Defines from the start of the Major Frame when the time window is
released in nanoseconds.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="PeriodicProcessingStart" type="xs:boolean" use="required">
            <xs:annotation>
                <xs:documentation>A point in a partition schedule aligned to the beginning of a partition's
window where the partition's periodic process scheduling is permitted to start.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="A653_MemoryRegionType">
        <xs:annotation>
```

```
            <xs:documentation>ARINC 653 Memory Region Type</xs:documentation>
        </xs:annotation>
        <xs:attribute name="Name" type="NameType" use="required">
            <xs:annotation>
                <xs:documentation>Name of the memory region</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Type" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>Type of the memory region</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Size" type="DecOrHexValueType" use="required">
            <xs:annotation>
                <xs:documentation>Size of the memory region in bytes</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Address" type="DecOrHexValueType">
            <xs:annotation>
                <xs:documentation>Start address of the memory region</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="AccessRights" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>Access right of the memory region</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="A653_MultiPartitionHMTableType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Multi-Partition HM table type</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="ErrorAction" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>List of errors and their corresponding recovery
actions.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="ErrorIdentifierRef" type="IdentifierValueType" use="required">
                        <xs:annotation>
                            <xs:documentation>Reference to Error Identifier as defined by the
list of A653_ErrorIdentifierType</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="ErrorLevel" type="MultiPartitionHMTableErrorLevelType"
use="required">
                        <xs:annotation>
                            <xs:documentation>Level of the error (MODULE or
PARTITION)</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="ModuleRecoveryAction"
type="ModuleLevelErrorRecoveryActionType" use="optional">
                        <xs:annotation>
                            <xs:documentation>Recovery action to take when error level is
MODULE (IGNORE, SHUTDOWN, RESET).</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="TableIdentifier" type="IdentifierValueType" use="optional">
            <xs:annotation>
                <xs:documentation>Optional identifier of the table. May be used to reference this table from a
partition configuration element.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="TableName" type="NameType" use="required">
            <xs:annotation>
                <xs:documentation>Name of the table. May be used to reference this table from a partition
configuration element.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="A653_PartitionHMTableType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Partition HM table Type</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="ErrorAction" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>List of errors and their corresponding recovery
actions.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="ErrorIdentifierRef" type="IdentifierValueType" use="required">
                        <xs:annotation>
                            <xs:documentation>Reference to the Error Identifier as defined by
the list of A653_ErrorIdentifierType; restricted to errors which can be raised by partitions.</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="ErrorLevel" type="PartitionHMTableErrorLevelType"
use="required">
                        <xs:annotation>
                            <xs:documentation>Level of the error (PARTITION or
PROCESS)</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="PartitionRecoveryAction"
type="PartitionLevelErrorRecoveryActionType" use="required">
                        <xs:annotation>
                            <xs:documentation>Recovery action to take when error level is
PARTITION or when Error Handler is not created (IGNORE, IDLE, WARM_RESTART, COLD_RESTART).</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="ErrorCode" type="ErrorCodeType" use="optional">
                        <xs:annotation>
                            <xs:documentation>Error code to be provided to Error Handler
process (DEADLINE_MISSED, APPLICATION_ERROR, NUMERIC_ERROR, ILLEGAL_REQUEST, STACK_OVERFLOW,
MEMORY_VIOLATION, HARDWARE_FAULT, POWER_FAIL).</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="TableIdentifier" type="IdentifierValueType" use="optional">
            <xs:annotation>
                <xs:documentation>Optional identifier of the table. May be used to reference this table from a
partition configuration element.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="TableName" type="NameType" use="required">
            <xs:annotation>
                <xs:documentation>Name of the table. May be used to reference this table from a partition
configuration element.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="MultiPartitionHMTableNameRef" type="NameType" use="required">
            <xs:annotation>
                <xs:documentation>Reference to the multi-partition HM table name</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="A653_ModuleHMTableType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Module HM table Type</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="ErrorAction" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>List of errors and their corresponding recovery
actions.</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:attribute name="ErrorIdentifierRef" type="IdentifierValueType" use="required">
                        <xs:annotation>
                            <xs:documentation>Reference to the Error Identifier as defined by
the list of A653_ErrorIdentifierType</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                    <xs:attribute name="ModuleRecoveryAction"
type="ModuleLevelErrorRecoveryActionType" use="required">
                        <xs:annotation>
                            <xs:documentation>Recovery action to take when this error
occurs (IGNORE, SHUTDOWN,RESET).</xs:documentation>
                        </xs:annotation>
                    </xs:attribute>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="StateIdentifier" type="IdentifierValueType" use="required">
            <xs:annotation>
                <xs:documentation>Module state identifier.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Description" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>Description of the state.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
    <xs:complexType name="A653_ErrorIdentifierType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Error Identifier Type</xs:documentation>
        </xs:annotation>
        <xs:attribute name="ErrorIdentifier" type="IdentifierValueType" use="required">
            <xs:annotation>
                <xs:documentation>Unique error identifier.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Description" type="xs:string" use="required">
            <xs:annotation>
                <xs:documentation>Description of the error.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
</xs:schema>
```

ARINC653TypesExtensible.xsd :

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:schema xmlns="ARINC653" xmlns:ar="ARINC653" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="ARINC653" elementFormDefault="qualified" attributeFormDefault="unqualified">
        <xs:simpleType name="ModuleLevelErrorRecoveryActionType">
                <xs:annotation>
                        <xs:documentation>Error recovery actions to take when module level errors occur.</xs:documentation>
                </xs:annotation>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="IGNORE"/>
                        <xs:enumeration value="SHUTDOWN"/>
                        <xs:enumeration value="RESET"/>
                        <xs:enumeration value=""/>
                </xs:restriction>
        </xs:simpleType>
</xs:schema>
```

APPENDIX I
ARINC 653 XML-SCHEMA TYPE USAGE EXAMPLES

This appendix is divided into four sections. The first section presents an XML-Schema example using the ARINC 653 XML-Schema types defined by the standard. This example includes both a graphical representation and the schema itself.

The second section gives an example of extending an ARINC 653 XML-Schema type.

The third section presents an example of a module HM configuration.

Finally, the fourth section gives an XML instantiation example using the XML-Schema created in the first section of this appendix.

I.1 XML-Schema Example

The schema presented in this appendix is relatively simple and can be used as the starting point to configure a full-fledge ARINX 653 module. The intent is to show how an O/S implementer may create an XML-Schema using the ARINC 653 types specified by this standard. This schema is not intended to be used as is by any O/S implementer.

XML Schema root element

Note: In this section, all words in italics refer to a type defined in the ARINC 653 XML-Schema type or in the XML-Schema example.

The XML schema root element is named MODULE and is shown in Figure 1**.**

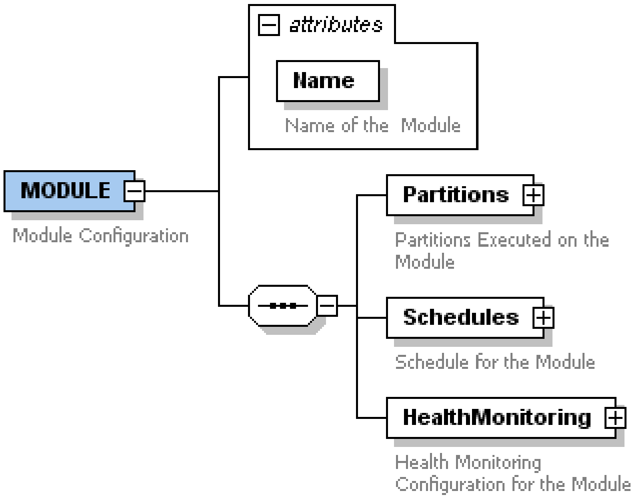


Figure 1 – XML Schema Root Element

The root element has one attribute: *Name*. This is the name of the module. Its type is *NameType*, which is defined in the ARINC 653-Schema type.

The root element has also a sequence of three elements: *Partitions*, *Schedules* and *HealthMonitoring*.

The *Partitions* element describes the partitions configuration in the module. Its type is *PartitionsType*, which is a new type created within the XML-schema and explained below.

The *Schedules* element describes the schedule configuration. Its type is *ScheduleType*. This is a newly created type within the XML-schema.

Finally the *HealthMonitoring* element represents the health-monitoring configuration for the module.

PartitionsType Complex Type

A graphical representation of the *PartitionType* complex type is shown in Figure 2**.**

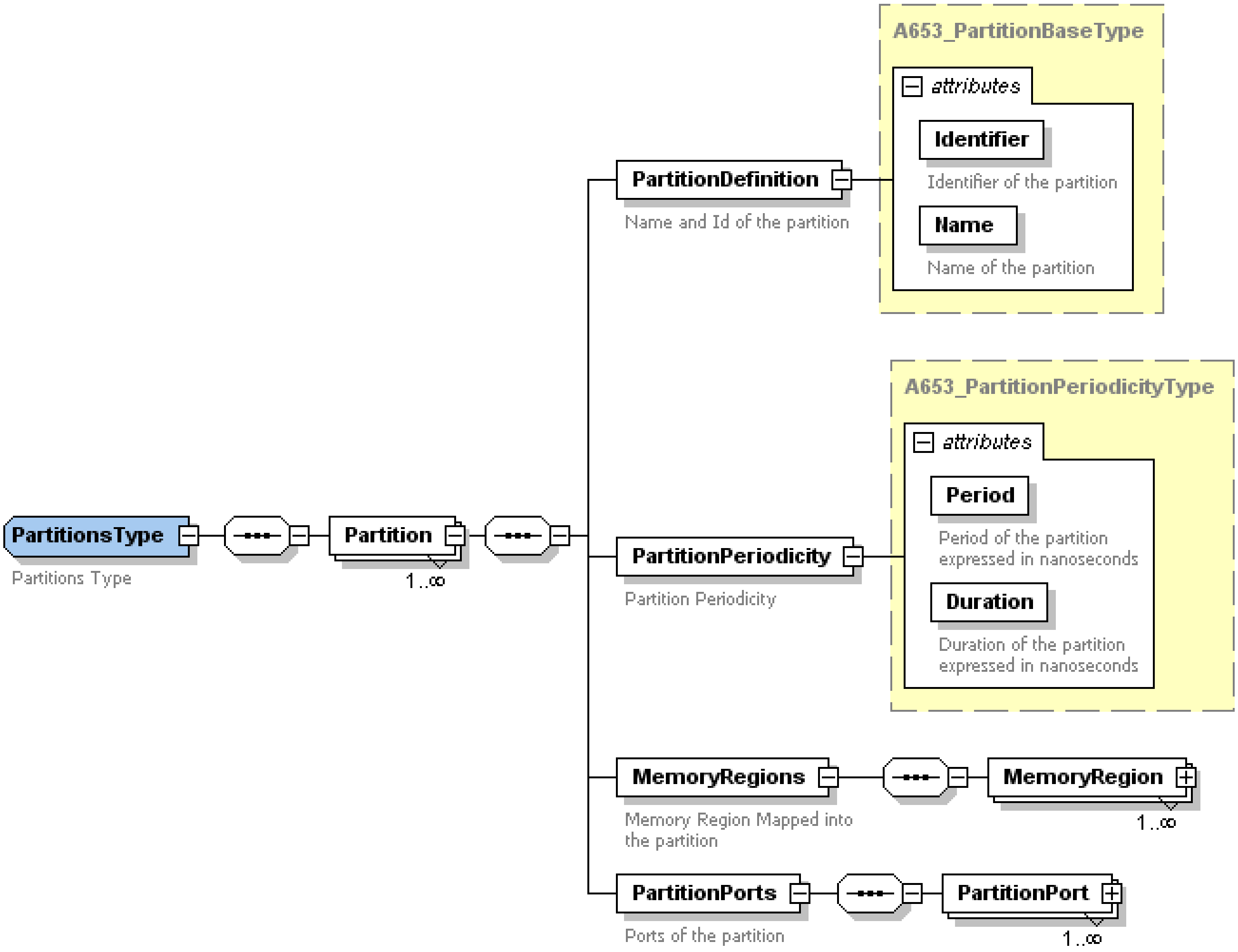


Figure 2 – PartitionsType Complex Type

The *PartitionsType* complex type is composed of a sequence of at least one element named *Partition*. This element *Partition* represents the configuration of one partition in the module. In the XML instantiation file, each partition will have its own instantiation of the element named *Partition*.

The element *Partition* is a sequence of four elements named: *PartitionDefinition*, *PartitionPeriodicity, MemoryRegions* and *PartitionPorts.*

The *PartitionDefinition* element defines the name and the identifier of the partition. Its type is *A653_PartitionBaseType*, which is defined in the ARINC 653 XML-Schema types.

The *PartitionPeriodicity* element defines the partition period and duration. Its type is *PA653_PartitionPeriodicityType*, which is defined in the ARINC 653 XML-Schema types.

The *MemoryRegions* element defines a sequence of at least one element named *MemoryRegion.* This element defines one memory region that is mapped to this partition by the OS. Each element defines one memory region that is allocated to the partition by the OS. This element's type is *A653_MemoryRegionType*, which is defined in the ARINC 653 XML-Schema types.

Finally, the *PartitionPorts* element defines the ARINC 653 ports accessible by the partition. At least one ARINC 653 port must be defined for each partition. The type of this element is *PortType*, which is a new type created within the XML-schema. This type is a selection between a queuing and a sampling port and is described below.

PortType Complex Type

Figure 3 is a graphical representation of the *PortType*.

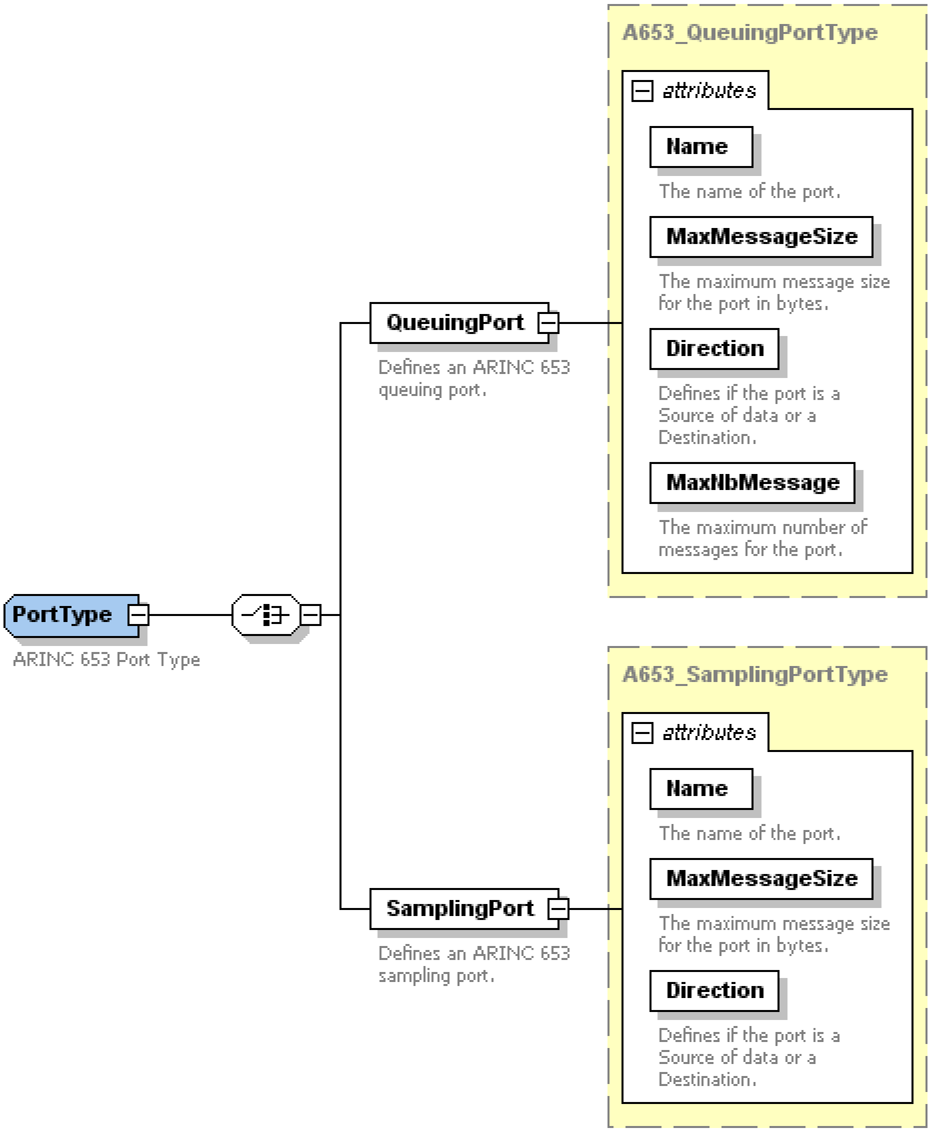

A653_QueuingPortType
attributes
Name
The name of the port.
MaxMessageSize
The maximum message size for the port in bytes.
Direction
Defines if the port is a Source of data or a Destination.
MaxNbMessage
The maximum number of messages for the port.
QueuingPort
Defines an ARINC 653 queuing port.
PortType
ARINC 653 Port Type
A653_SamplingPortType
attributes
Name
The name of the port.
MaxMessageSize
The maximum message size for the port in bytes.
Direction
Defines if the port is a Source of data or a Destination.
SamplingPort
Defines an ARINC 653 sampling port.

Figure 3 – PortType Complex Type

The element *QueuingPort* defines a queuing port configuration and is based on the *A653_QueuingPortType*, which is defined in the ARINC 653 XML-Schema types.

The element *SamplingPort* defines a sampling port configuration and is based on the *A653_SamplingPortType,* which is also defined in the ARINC 653 XML-Schema types.

ScheduleType Complex Type

The *ScheduleType* complex type defines the schedule to be executed by the ARINC 653 scheduler. This schedule defines a series of time window where, within each window, a partition is executed.

Figure 4 **shows a graphical representation of the *ScheduleType.***

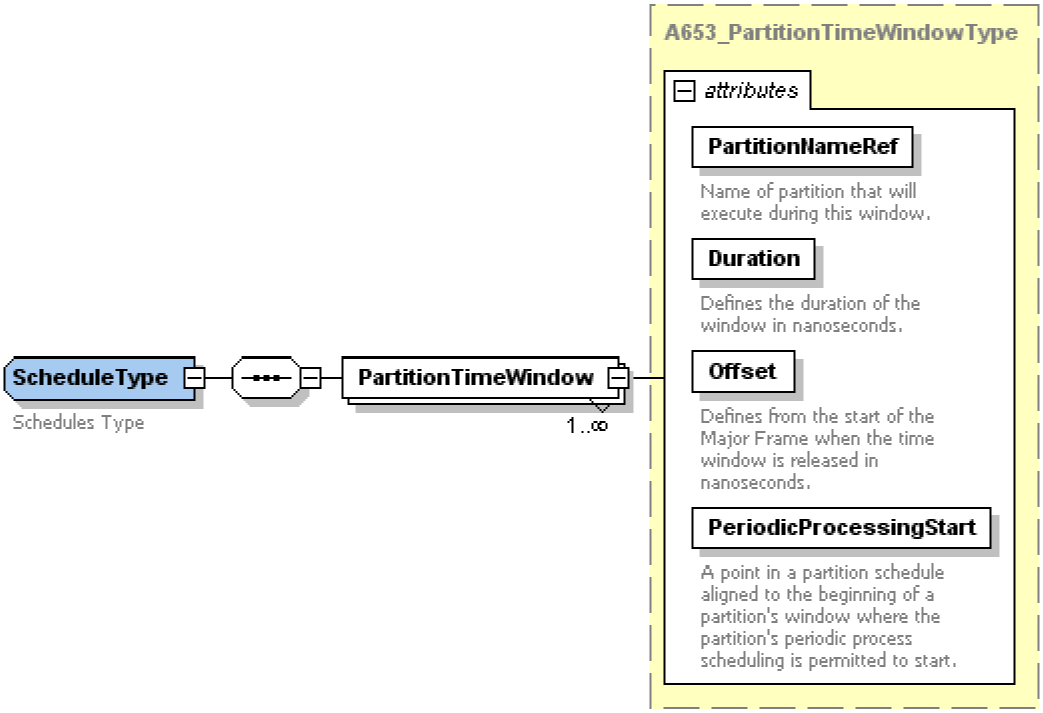

A653_PartitionTimeWindowType
attributes
PartitionNameRef
Name of partition that will execute during this window.
Duration
Defines the duration of the window in nanoseconds.
Offset
Defines from the start of the Major Frame when the time window is released in nanoseconds.
PeriodicProcessingStart
A point in a partition schedule aligned to the beginning of a partition's window where the partition's periodic process scheduling is permitted to start.
ScheduleType
Schedules Type
PartitionTimeWindow
1..∞

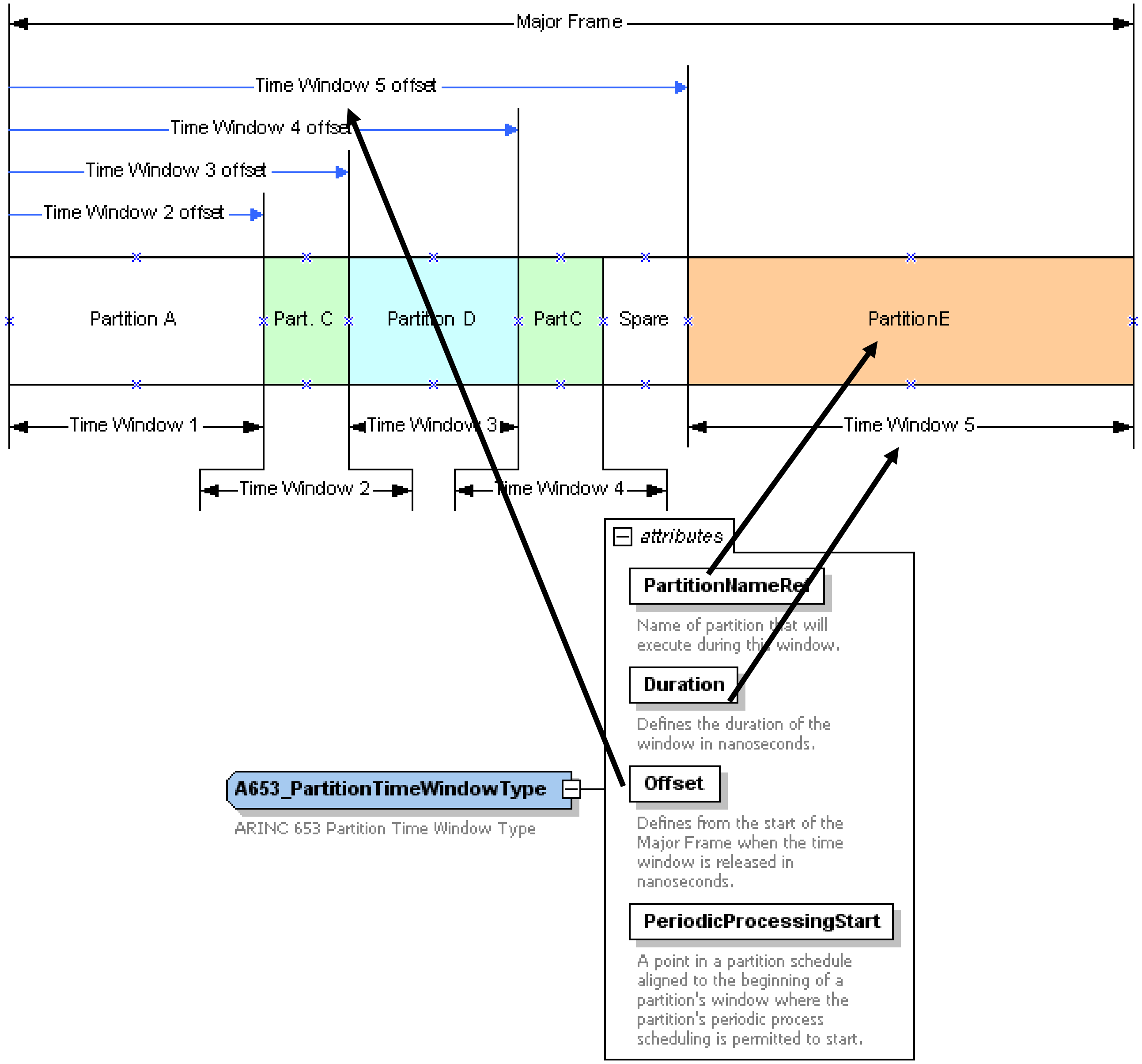
Figure 4 – ScheduleType Complex Type

The *ScheduleType* complex type is a sequence of at least one element named *PartitionTimeWindow*. The type of the element is *A653_PartitionTimeWindowType*, which is defined in the ARINC 653 XML-Schema types.

In the XML instantiation file, each partition time window will be configured using this element.

Figure 5 **shows a schedule composed of five time windows and how the *PartitionTimeWindowType* can be used to create such schedule.**

APPENDIX I
ARINC 653 XML-SCHEMA TYPE USAGE EXAMPLES



Name of partition that will execute during this window.

Defines the duration of the window in nanoseconds.

Defines from the start of the Major Frame when the time window is released in nanoseconds.

A point in a partition schedule aligned to the beginning of a partition's window where the partition's periodic process scheduling is permitted to start.

ARINC 653 Partition Time Window Type

Figure 5 – Example of A653_PartitionTimeWindowType Usage

HealthMonitoringType Complex Type

Figure 6 shows a graphical representation of the *HealthMonitoringType.*

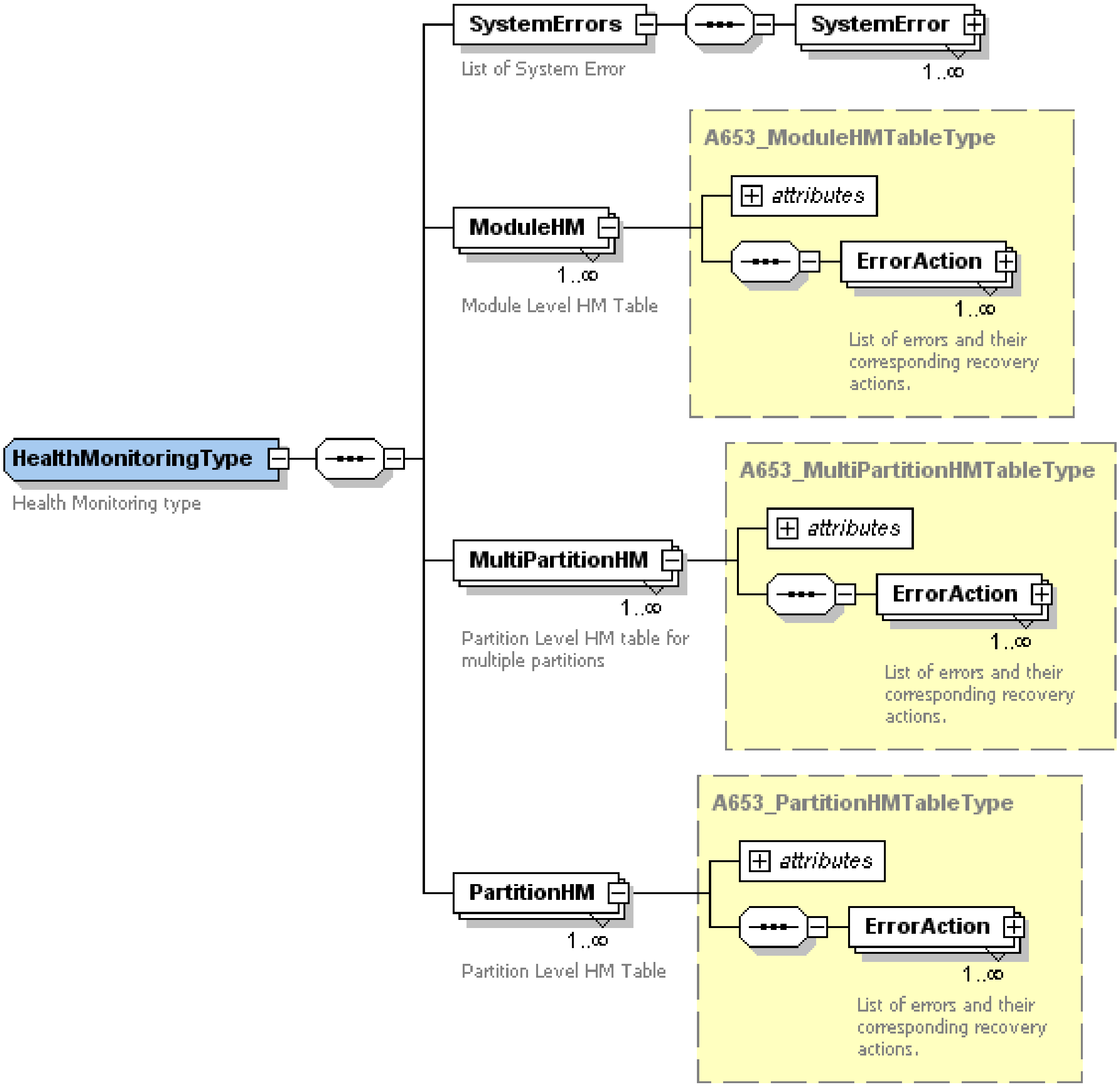


Figure 6 – HealthMonitoringType Complex Type

The *HealthMonitoringType* is a sequence of four elements: *SystemError, ModuleHM*, *MultiPartitionHM* and *PartitionHM*.

The *SystemErrors* element describes all the possible errors in the system. Its type is composed of the *ErrorIdentifierType*, which is located in the ARINC 653 XML-Schema types.

The *ModuleHM* element describes the Module-level health-monitoring configuration; it is based on the *ModuleHMTableType* located in the ARINC 653 XML-Schema types.

The *MultiPartitionHM* element describes the health-monitoring configuration that can be used for multiple partitions at the same time. Its type is *MultiPartitionHMTableType*, which is located in the ARINC 653 XML-Schema types.

The *PartitionHM* element describes the partition health-monitoring configuration. This element is based on the *PartitionHMTableType* located in the ARINC 653 XML-Schema types.

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="ARINC653" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="ARINC653"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:include schemaLocation="ARINC653Types.xsd"/>
    <xs:complexType name="PortType">
        <xs:annotation>
            <xs:documentation>ARINC 653 Port Type</xs:documentation>
        </xs:annotation>
        <xs:choice>
            <xs:element name="QueuingPort" type="A653_QueuingPortType">
                <xs:annotation>
                    <xs:documentation>Defines an ARINC 653 queuing port.</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="SamplingPort" type="A653_SamplingPortType">
                <xs:annotation>
                    <xs:documentation>Defines an ARINC 653 sampling port.</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="PartitionsType">
        <xs:annotation>
            <xs:documentation>Partitions Type</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="Partition" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="PartitionDefinition"
type="A653_PartitionBaseType">
                            <xs:annotation>
                                <xs:documentation>Name and Id of the
partition</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="PartitionPeriodicity"
type="A653_PartitionPeriodicityType">
                            <xs:annotation>
                                <xs:documentation>Partition
Periodicity</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="MemoryRegions">
                            <xs:annotation>
                                <xs:documentation>Memory Region Mapped into
the partition</xs:documentation>
                            </xs:annotation>
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="MemoryRegion"
type="A653_MemoryRegionType" maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="PartitionPorts">
                            <xs:annotation>
                                <xs:documentation>Ports of the
partition</xs:documentation>
                            </xs:annotation>
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="PartitionPort"
type="PortType" maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ScheduleType">
        <xs:annotation>
            <xs:documentation>Schedules Type</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="PartitionTimeWindow" type="A653_PartitionTimeWindowType"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="HealthMonitoringType">
        <xs:annotation>
            <xs:documentation>Health Monitoring type</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="SystemErrors">
                <xs:annotation>
                    <xs:documentation>List of System Error</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="SystemError" type="A653_ErrorIdentifierType"
maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="ModuleHM" type="A653_ModuleHMTableType" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>Module Level HM Table</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="MultiPartitionHM" type="A653_MultiPartitionHMTableType"
maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>Partition Level HM table for multiple partitions
</xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="PartitionHM" type="A653_PartitionHMTableType" maxOccurs="unbounded">
                <xs:annotation>
                    <xs:documentation>Partition Level HM Table</xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="MODULE">
        <xs:annotation>
            <xs:documentation>Module Configuration</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Partitions" type="PartitionsType">
                    <xs:annotation>
                        <xs:documentation>Partitions Executed on the
Module</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="Schedules" type="ScheduleType">
                    <xs:annotation>
                        <xs:documentation>Schedule for the Module</xs:documentation>
                    </xs:annotation>
                </xs:element>
                <xs:element name="HealthMonitoring" type="HealthMonitoringType">
                    <xs:annotation>
                        <xs:documentation>Health Monitoring Configuration for the
Module</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="Name" type="NameType" use="required">
                <xs:annotation>
                    <xs:documentation>Name of the  Module</xs:documentation>
```

```
                                        </xs:annotation>
                                </xs:attribute>
                        </xs:complexType>
                </xs:element>
</xs:schema>
```

I.2 ARINC 653 XML-Schema Extension Example

This section is an example of extending an ARINC 653 XML-schema type. In this example, the goal is to add cache configurations to the ARINC 653 memory region type.

This extension is divided into two parts:

1. Create an XML-schema simple type that list the cache setting supported by the OS.
2. Extend the A653_MemoryRegionType.

A simple type that lists cache setting supported by the OS is added to the module configuration XML-schema. **This simple type is name *cacheSettingType,* and its type is xs:string, which is an XML-schema predefine type.**

The supported cache settings are:

- **Cache_Off**
- **Cache_WriteThrough**
- **Cache_CopyBack**

The XML-schema code for this new type is presented in Figure 7**.**

```
<xs:simpleType name="cacheSettingType">
        <xs:restriction base="xs:string">
                <xs:enumeration value="Cache_Off"/>
                <xs:enumeration value="Cache_WriteThrough"/>
                <xs:enumeration value="Cache_CopyBack"/>
        </xs:restriction>

</xs:simpleType>
```

Figure 7 – cacheSettingType Simple Type

With the first part done, it is time to extend the *A653_MemoryRegionType*. Since it is part of ARINC653Types.xml file, this type is one of the non-modifiable types defined by this standard. The extension must be done somewhere else. For the example, the extension is done in the XML-schema.

In the XML-schema file, a new complex type is created named *memorySectionType*. Its base type is *A653_MemoryRegionType* and has an attribute named *cacheSetting.* The type for this attribute is *cacheSettingType*, which is defined above.

This new complex type is now created. Figure 8 **is a graphical presentation of this new type.**

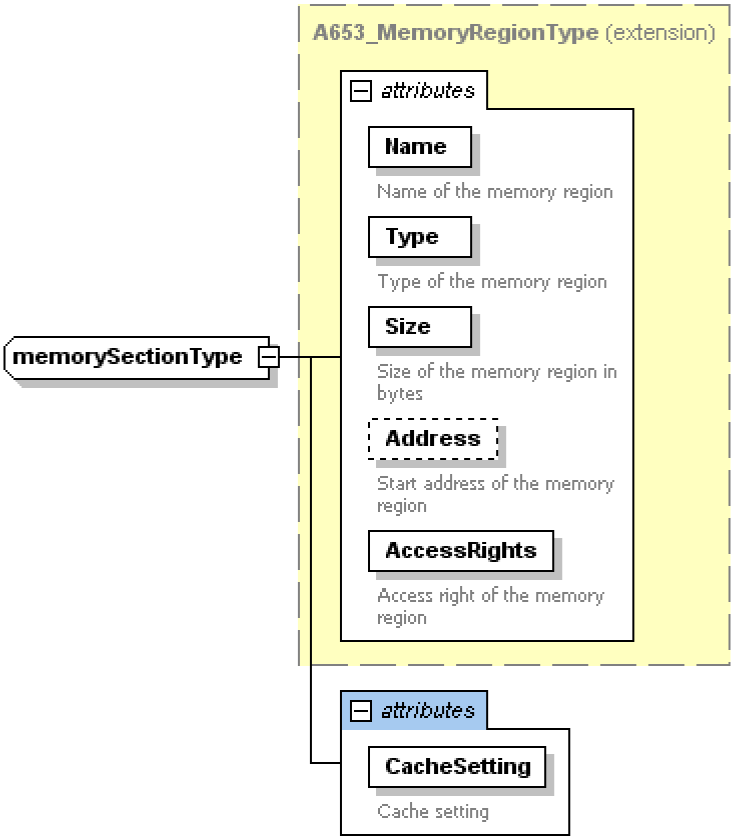

A653_MemoryRegionType (extension)
attributes
Name
Name of the memory region
Type
Type of the memory region
Size
Size of the memory region in bytes
Address
Start address of the memory region
AccessRights
Access right of the memory region
memorySectionType
attributes
CacheSetting
Cache setting

Figure 8 – memorySectionType Complex Type

Figure 9 is the XML-schema code for this new type.

```
<xs:complexType name="memorySection">
        <xs:complexContent>
                <xs:extension base="A653_MemoryRegionType">
                        <xs:attribute name="CacheSetting" type="cacheSettingType" use="required">
                                <xs:annotation>
                                        <xs:documentation>Cache setting</xs:documentation>
                                </xs:annotation>
                        </xs:attribute>
                </xs:extension>
        </xs:complexContent>

</xs:complexType>
```

Figure 9 – memorySectionType XML-Schema

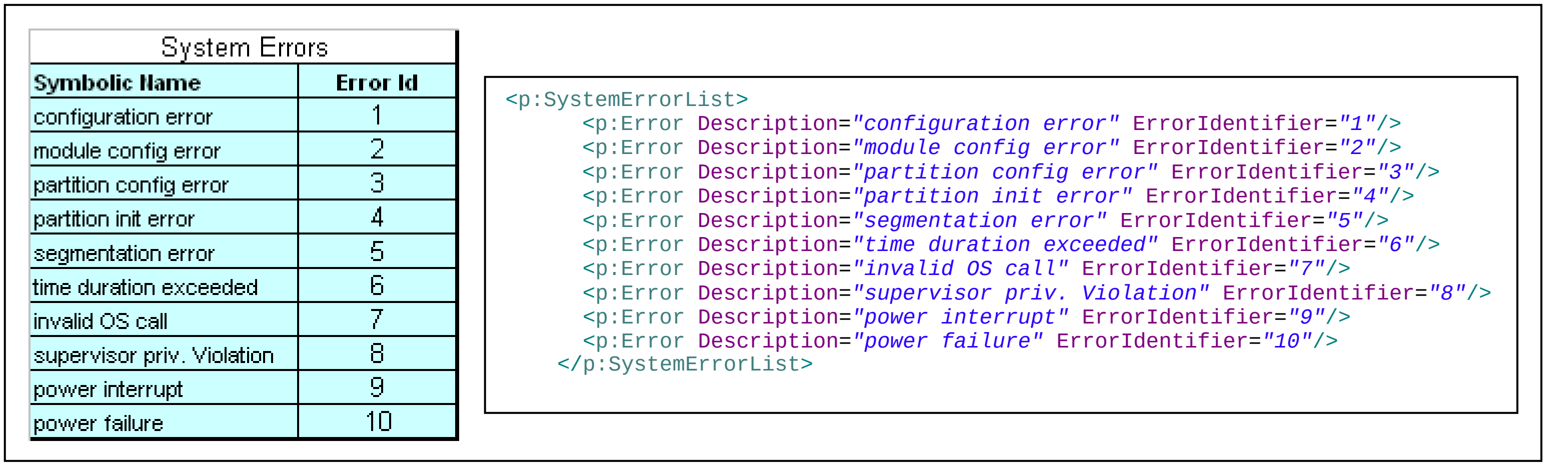
I.3 Health Monitoring Configuration Example

This section presents an ARINC 653 HM configuration example. This example uses the XLM-schema presented in Section I.2.

The module contains the following partitions:

- Two application partitions named “application 1” and “application 2”.
- One system partition named “system partition 1”.

The system error are defined in the *SystemErrorList* element and presented in Figure 10.



System Errors	
Symbolic Name	**Error Id**
configuration error	1
module config error	2
partition config error	3
partition init error	4
segmentation error	5
time duration exceeded	6
invalid OS call	7
supervisor priv. Violation	8
power interrupt	9
power failure	10

```
<p:SystemErrorList>
      <p:Error Description="configuration error" ErrorIdentifier="1"/>
      <p:Error Description="module config error" ErrorIdentifier="2"/>
      <p:Error Description="partition config error" ErrorIdentifier="3"/>
      <p:Error Description="partition init error" ErrorIdentifier="4"/>
      <p:Error Description="segmentation error" ErrorIdentifier="5"/>
      <p:Error Description="time duration exceeded" ErrorIdentifier="6"/>
      <p:Error Description="invalid OS call" ErrorIdentifier="7"/>
      <p:Error Description="supervisor priv. Violation" ErrorIdentifier="8"/>
      <p:Error Description="power interrupt" ErrorIdentifier="9"/>
      <p:Error Description="power failure" ErrorIdentifier="10"/>
    </p:SystemErrorList>
```

Figure 10 – System Error Configuration Example

Some errors are synchronous to partition execution, and others are not. Therefore, an error should be allocated to:

- The module HM configuration, when this error is possible in the internal module states.
- The multi-partition and partition HM configuration, when this error happens on the partition windows.

The Module HM configuration is composed of several entries of *A653_ModuleHMTableType*, each entry describes the error management for a system state, describing the errors that are possible at that state. Figure 11 presents this configuration.

Detected Error		module init	module function	partition switch
Symbolic Name	**Error Id**	state 1	state 2	state 3
configuration error	1	SHUTDOWN		
module config error	2	RESET		
segmentation error	5	SHUTDOWN	SHUTDOWN	SHUTDOWN
time duration exceeded	6	IGNORE	RESET	RESET
invalid OS call	7	SHUTDOWN	SHUTDOWN	SHUTDOWN

```
 <p:ModuleHm Description="Module init" StateIdentifier="1">
  <p:ErrorAction ErrorIdentifierRef="1" ModuleRecoveryAction="SHUTDOWN"/>
  <p:ErrorAction ErrorIdentifierRef="2" ModuleRecoveryAction="RESET" />
     <p:ErrorAction ErrorIdentifierRef="5" ModuleRecoveryAction="SHUTDOWN" />
     <p:ErrorAction ErrorIdentifierRef="6" ModuleRecoveryAction="IGNORE" />
     <p:ErrorAction ErrorIdentifierRef="7" ModuleRecoveryAction="SHUTDOWN" />
</p:ModuleHm>

 <p:ModuleHm Description="Module function" StateIdentifier="2">
     <p:ErrorAction ErrorIdentifierRef="5" ModuleRecoveryAction="SHUTDOWN" />
     <p:ErrorAction ErrorIdentifierRef="6" ModuleRecoveryAction="RESET" />
     <p:ErrorAction ErrorIdentifierRef="7" ModuleRecoveryAction="SHUTDOWN" />
</p:ModuleHm>
<p:ModuleHm Description="Partition switch" StateIdentifier="3">
     <p:ErrorAction ErrorIdentifierRef="5" ModuleRecoveryAction="SHUTDOWN" />
     <p:ErrorAction ErrorIdentifierRef="6" ModuleRecoveryAction="RESET" />
     <p:ErrorAction ErrorIdentifierRef="7" ModuleRecoveryAction="SHUTDOWN" />

</p:ModuleHm>
```

Figure 11 – Module Error Management Configuration Example

All errors allocated to the partitions are first described in the multi-partition tables. In these tables, the system integrator can control the scope of the errors and allocate the errors that are handled by the partitions. In our example, we have two multi-partition tables: one for the application partitions and another for the system partition.

The partitions error management is described by this configuration presented in Figure 12**.**

A default recovery action is defined at integration level for error 8 and 9; the other recovery actions have to be defined by the application developers.

Application Multipartition Table id=1

Detected Error		
Error Id	**Error Level**	**Recovery Action**
3	PARTITION	
4	PARTITION	
5	PARTITION	
6	PARTITION	
7	PARTITION	
8	MODULE	SHUTDOWN
9	MODULE	IGNORE
10	PARTITION	

```
<p:MultiPartitionHM TableIdentifier="1" TableName="Application MultiPartition
Table">
    <p:ErrorAction ErrorIdentifierRef="3" ErrorLevel="PARTITION"/>
    <p:ErrorAction ErrorIdentifierRef="4" ErrorLevel="PARTITION" />
    <p:ErrorAction ErrorIdentifierRef="5" ErrorLevel="PARTITION" />
    <p:ErrorAction ErrorIdentifierRef="6" ErrorLevel="PARTITION" />
    <p:ErrorAction ErrorIdentifierRef="7" ErrorLevel="PARTITION" />
    <p:ErrorAction ErrorIdentifierRef="8" ErrorLevel="MODULE"
                   ModuleRecoveryAction="SHUTDOWN"  />
    <p:ErrorAction ErrorIdentifierRef="9" ErrorLevel="MODULE"
                   ModuleRecoveryAction="IGNORE" />
        <p:ErrorAction ErrorIdentifierRef="10" ErrorLevel="PARTITION"/>
</p:MultiPartitionHM>
```

Figure 12 – Multi-Partition Configuration Example 1

Regarding the system partition, the error management is not the same. Its configuration is presented in Figure 13.

The system partitions are needed for the module to be operational, meaning that errors have mainly a module impact. Errors 5 and 10 are handled at partition level.

System Multipartition Table id=2

Detected Error		
Error Id	**Error Level**	**Recovery Action**
3	MODULE	SHUTDOWN
4	MODULE	SHUTDOWN
5	PARTITION	
6	MODULE	RESET
7	MODULE	RESET
8	MODULE	SHUTDOWN
9	MODULE	IGNORE
10	PARTITION	

```
<p:MultiPartitionHM TableIdentifier="2" TableName="System MultiPartition table">
    <p:ErrorAction ErrorIdentifierRef="3" ErrorLevel="MODULE"
                  ModuleRecoveryAction="SHUTDOWN" />
    <p:ErrorAction ErrorIdentifierRef="4" ErrorLevel="MODULE"
                   ModuleRecoveryAction="SHUTDOWN" />
    <p:ErrorAction ErrorIdentifierRef="5" ErrorLevel="PARTITION" />
    <p:ErrorAction ErrorIdentifierRef="6" ErrorLevel="MODULE"
                  ModuleRecoveryAction="RESET" />
    <p:ErrorAction ErrorIdentifierRef="7" ErrorLevel="MODULE"
                  ModuleRecoveryAction="RESET" />
    <p:ErrorAction ErrorIdentifierRef="8" ErrorLevel="MODULE"
                  ModuleRecoveryAction="SHUTDOWN" />
    <p:ErrorAction ErrorIdentifierRef="9" ErrorLevel="MODULE"
                   ModuleRecoveryAction="IGNORE" />
    <p:ErrorAction ErrorIdentifierRef="10" ErrorLevel="PARTITION"/>
</p:MultiPartitionHM>
```

Figure 13 – Multi-Partition Configuration Example 2

The partition HM configuration describes all partition level errors (in relation with the multi-partition configuration). As described in Section 2.4, a partition manages the error at partition level or at process level.

Figure 14 is the application partition HM configuration.

APPENDIX I
ARINC 653 XML-SCHEMA TYPE USAGE EXAMPLES

The partition 1 is managing errors 5, 6, 7, 10 at process level.

Partition 1 **Multi partition 1**

Detected Error			
Error Id	**Error Level**	Recovery Action	ErrorCode
3	PARTITION	IDLE	
4	PARTITION	COLD_RESTART	
5	PROCESS	WARM_RESTART	MEMORY_VIOLATION
6	PROCESS	COLD_RESTART	DEADLINE_MISSED
7	PROCESS	WARM_RESTART	ILLEGAL_REQUEST
10	PROCESS	IDLE	POWER_FAIL

```
<p:PartitionHM TableName="HM Table application 1"    MultiParitionHMTableNameRef="1">
        <p:ErrorAction ErrorIdentifierRef="3"
                PartitionRecoveryAction="IDLE" ErrorLevel="PARTITION" />
        <p:ErrorAction ErrorIdentifierRef="4"
                PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PARTITION" />
        <p:ErrorAction ErrorIdentifierRef="5"
                PartitionRecoveryAction="WARM_RESTART" ErrorLevel="PROCESS"
                ErrorCode="MEMORY_VIOLATION"></p:ErrorAction>
        <p:ErrorAction ErrorIdentifierRef="6"
                PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PROCESS"
                ErrorCode="DEADLINE_MISSED" />
        <p:ErrorAction ErrorIdentifierRef="7"
                PartitionRecoveryAction="WARM_RESTART" ErrorLevel="PROCESS"
                ErrorCode="ILLEGAL_REQUEST" />
        <p:ErrorAction ErrorIdentifierRef="10"
                PartitionRecoveryAction="IDLE" ErrorLevel="PROCESS"
                ErrorCode="POWER_FAIL" />

 </p:PartitionHM>
```

For partition 2 the same configuration except for the power supply management (error 10).

Partition 2 **Multi partition 1**

Detected Error			
Error Id	**Error Level**	Recovery Action	ErrorCode
3	PARTITION	IDLE	
4	PARTITION	COLD_RESTART	
5	PROCESS	WARM_RESTART	MEMORY_VIOLATION
6	PROCESS	COLD_RESTART	DEADLINE_MISSED
7	PROCESS	WARM_RESTART	ILLEGAL_REQUEST
10	PARTITION	IDLE	

```
<p:PartitionHM TableName="HM Table application 2" MultiParitionHMTableNameRef="1">
        <p:ErrorAction ErrorIdentifierRef="3"
                PartitionRecoveryAction="IDLE" ErrorLevel="PARTITION" />
        <p:ErrorAction ErrorIdentifierRef="4"
                PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PARTITION" />
        <p:ErrorAction ErrorIdentifierRef="5"
                PartitionRecoveryAction="WARM_RESTART" ErrorLevel="PROCESS"
                ErrorCode="MEMORY_VIOLATION" />
        <p:ErrorAction ErrorIdentifierRef="6"
                PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PROCESS"
                ErrorCode="DEADLINE_MISSED" />
        <p:ErrorAction ErrorIdentifierRef="7"
                PartitionRecoveryAction="WARM_RESTART" ErrorLevel="PROCESS"
                ErrorCode="ILLEGAL_REQUEST" />
        <p:ErrorAction ErrorIdentifierRef="10"
                PartitionRecoveryAction="IDLE" ErrorLevel="PARTITION" />
```

Figure 14 – Partition HM Configuration Example 1

Figure 15 is the system partition HM configuration.

This system partition manage error 10 at process level.

Partition 3 **Multi partition 2**

Detected Error			
Error Id	**Error Level**	Recovery Action	ErrorCode
5	PARTITION	COLD_RESTART	
10	PROCESS	IDLE	POWER_FAIL

```
<p:PartitionHM TableName="HM Table for system partition1" MultiParitionHMTableNameRef="2">
     <p:ErrorAction ErrorIdentifierRef="5" PartitionRecoveryAction="COLD_RESTART"
            ErrorLevel="PARTITION" />
     <p:ErrorAction ErrorIdentifierRef="10"
            PartitionRecoveryAction="IDLE" ErrorLevel="PROCESS"
            ErrorCode="POWER_FAIL" />
</p:PartitionHM>
```

Figure 15 – Partition HM Configuration Example 2

APPENDIX I
ARINC 653 XML-SCHEMA TYPE USAGE EXAMPLES

I.4 XML Instance file

The instance file shown in this Appendix is built from the example shown in Figure 16 **and Figure** 17.

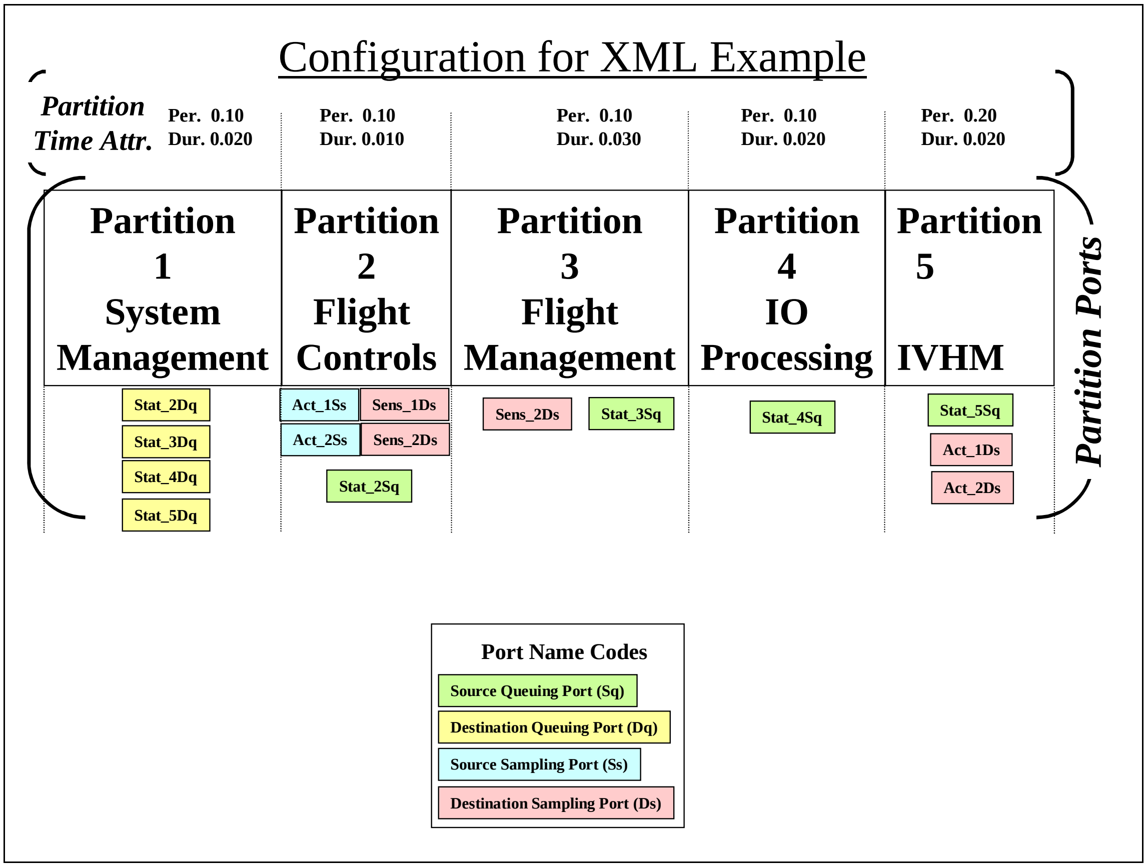


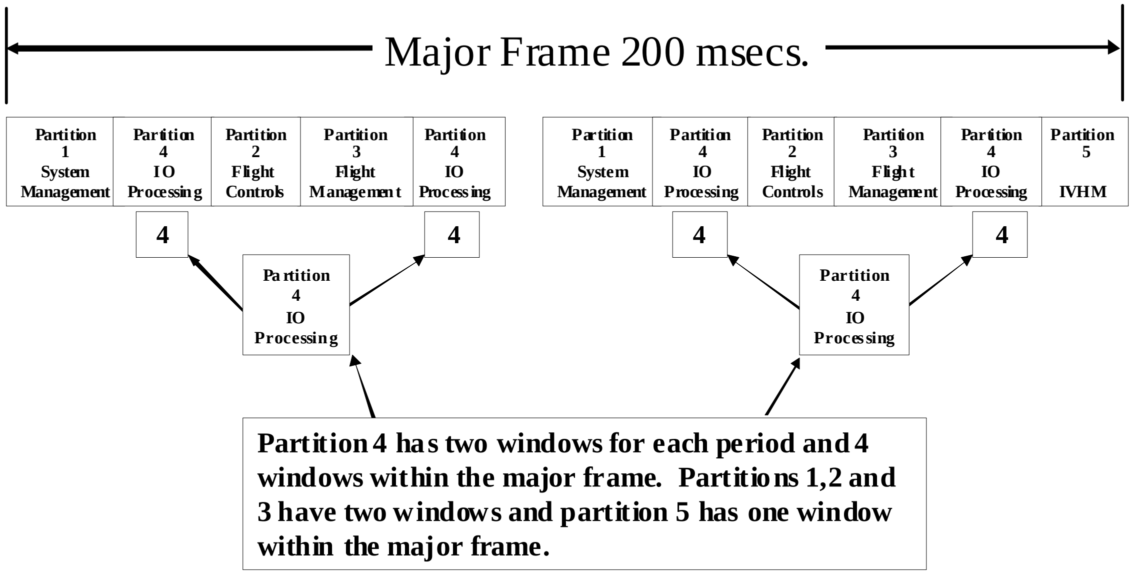
Figure 16 – Configuration for XML Example

Figure 16 shows a five-partition module. The time attributes of the **partitions** and the sampling and **queuing ports of the module are shown in the figure and are** completely defined in the XML instance.

The allocation of these five partitions to partition **time** windows is shown in **Figure** 17. Since the lowest frequency partition defines the major frame rate**, the duration of the major frame is** 200 milliseconds. The window requirements would define the start time (offset) of each window and its duration for the entire major frame.

Partition Window Requirements for XML Example

Window ID	1.1	4.1	2.1	3.1	4.2	1.2	4.3	2.2	3.2	4.4	5.1
Partition	P1	P4	P2	P3	P4	P1	P4	P2	P3	P4	P5
window offset	0.000	0.020	0.030	0.040	0.070	0.100	0.120	0.130	0.140	0.170	0.180
window duration	0.020	0.010	0.010	0.030	0.010	0.020	0.010	0.010	0.030	0.010	0.020


Major Frame 200 msecs.
Partition 1 System Management
Partition 4 IO Processing
Partition 2 Flight Controls
Partition 3 Flight Management
Partition 4 IO Processing
Partition 1 System Management
Partition 4 IO Processing
Partition 2 Flight Controls
Partition 3 Flight Management
Partition 4 IO Processing
Partition 5 IVHM
4
4
4
4
Partition 4 IO Processing
Partition 4 IO Processing

Partition 4 has two windows for each period and 4 windows within the major frame. Partitions 1,2 and 3 have two windows and partition 5 has one window within the major frame.

Figure 17 – Partition Window Time Requirements

The XML Instance example for these figures is as follows:

APPENDIX I
ARINC 653 XML-SCHEMA TYPE USAGE EXAMPLES

```
<?xml version="1.0" encoding="UTF-8"?>
<ar:MODULE Name="ARINC 653 Module" xsi:schemaLocation="ARINC653 ModuleExample.xsd"
xmlns:ar="ARINC653" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <ar:Partitions>
                <ar:Partition>
                        <ar:PartitionDefinition Name="systemManagement" Identifier="1"/>
                        <ar:PartitionPeriodicity Duration="20000000" Period="100000000"/>
                        <ar:MemoryRegions>
                                <ar:MemoryRegion Type="RAM" Size="1048576" Name="mainMemory"
AccessRights="READ_WRITE"/>
                                <ar:MemoryRegion Type="Flash" Size="524288" Name="Flash"
AccessRights="READ_ONLY"/>
                        </ar:MemoryRegions>
                        <ar:PartitionPorts>
                                <ar:PartitionPort>
                                        <ar:QueuingPort MaxMessageSize="30" Name="Stat_2Dq"
MaxNbMessage="30" Direction="DESTINATION"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:QueuingPort MaxMessageSize="30" Name="Stat_3Dq"
MaxNbMessage="30" Direction="DESTINATION"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:QueuingPort MaxMessageSize="30" Name="Stat_4Dq"
MaxNbMessage="30" Direction="DESTINATION"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:QueuingPort MaxMessageSize="30" Name="Stat_5Dq"
MaxNbMessage="30" Direction="DESTINATION"/>
                                </ar:PartitionPort>
                        </ar:PartitionPorts>
                </ar:Partition>
                <ar:Partition>
                        <ar:PartitionDefinition Name="flightControls" Identifier="2"/>
                        <ar:PartitionPeriodicity Duration="10000000" Period="100000000"/>
                        <ar:MemoryRegions>
                                <ar:MemoryRegion Type="RAM" Size="1048576" Name="mainMemory"
AccessRights="READ_WRITE"/>
                                <ar:MemoryRegion Type="Flash" Size="524288" Name="Flash"
AccessRights="READ_ONLY"/>
                        </ar:MemoryRegions>
                        <ar:PartitionPorts>
                                <ar:PartitionPort>
                                        <ar:SamplingPort MaxMessageSize="20" Name="Act_1Ss"
Direction="SOURCE"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:SamplingPort MaxMessageSize="20" Name="Act_2Ss"
Direction="SOURCE"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:SamplingPort MaxMessageSize="40" Name="Sens_1Ds"
Direction="DESTINATION"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:SamplingPort MaxMessageSize="40" Name="Sens_2Ds"
Direction="DESTINATION"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:QueuingPort MaxMessageSize="30" Name="Stat_2Sq"
MaxNbMessage="30" Direction="SOURCE"/>
                                </ar:PartitionPort>
                        </ar:PartitionPorts>
                </ar:Partition>
                <ar:Partition>
                        <ar:PartitionDefinition Name="flightManagement" Identifier="3"/>
                        <ar:PartitionPeriodicity Duration="30000000" Period="100000000"/>
                        <ar:MemoryRegions>
                                <ar:MemoryRegion Type="RAM" Size="1048576" Name="mainMemory"
AccessRights="READ_WRITE"/>
                                <ar:MemoryRegion Type="Flash" Size="524288" Name="Flash"
AccessRights="READ_ONLY"/>
                        </ar:MemoryRegions>
                        <ar:PartitionPorts>
```

```
                                <ar:PartitionPort>
                                        <ar:SamplingPort MaxMessageSize="40" Name="Sens_2Ds"
Direction="DESTINATION"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:QueuingPort MaxMessageSize="30" Name="Stat_3Sq"
MaxNbMessage="30" Direction="SOURCE"/>
                                </ar:PartitionPort>
                        </ar:PartitionPorts>
                </ar:Partition>
                <ar:Partition>
                        <ar:PartitionDefinition Name="IOProcessing" Identifier="4"/>
                        <ar:PartitionPeriodicity Duration="20000000" Period="100000000"/>
                        <ar:MemoryRegions>
                                <ar:MemoryRegion Type="RAM" Size="1048576" Name="mainMemory"
AccessRights="READ_WRITE"/>
                                <ar:MemoryRegion Type="Flash" Size="524288" Name="Flash"
AccessRights="READ_ONLY"/>
                                <ar:MemoryRegion Type="Input/Ouput" Size="524288" Name="IORegion"
AccessRights="READ_WRITE"/>
                        </ar:MemoryRegions>
                        <ar:PartitionPorts>
                                <ar:PartitionPort>
                                        <ar:QueuingPort MaxMessageSize="30" Name="Stat_4Sq"
MaxNbMessage="30" Direction="SOURCE"/>
                                </ar:PartitionPort>
                        </ar:PartitionPorts>
                </ar:Partition>
                <ar:Partition>
                        <ar:PartitionDefinition Name="IHVM" Identifier="5"/>
                        <ar:PartitionPeriodicity Duration="20000000" Period="200000000"/>
                        <ar:MemoryRegions>
                                <ar:MemoryRegion Type="RAM" Size="1048576" Name="mainMemory"
AccessRights="READ_WRITE"/>
                                <ar:MemoryRegion Type="Flash" Size="524288" Name="Flash"
AccessRights="READ_ONLY"/>
                        </ar:MemoryRegions>
                        <ar:PartitionPorts>
                                <ar:PartitionPort>
                                        <ar:SamplingPort MaxMessageSize="20" Name="Act_1Ds"
Direction="DESTINATION"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:SamplingPort MaxMessageSize="20" Name="Act_2Ds"
Direction="DESTINATION"/>
                                </ar:PartitionPort>
                                <ar:PartitionPort>
                                        <ar:QueuingPort MaxMessageSize="30" Name="Stat_5Sq"
MaxNbMessage="30" Direction="SOURCE"/>
                                </ar:PartitionPort>
                        </ar:PartitionPorts>
                </ar:Partition>
        </ar:Partitions>
        <ar:Schedules>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="20000000"
PartitionNameRef="systemManagement" Offset="0"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="10000000"
PartitionNameRef="IOProcessing" Offset="20000000"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="10000000"
PartitionNameRef="flightControl" Offset="30000000"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="30000000"
PartitionNameRef="flightManagement" Offset="40000000"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="10000000"
PartitionNameRef="IOProcessing" Offset="70000000"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="20000000"
PartitionNameRef="systemManagement" Offset="100000000"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="10000000"
PartitionNameRef="IOProcessing" Offset="120000000"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="10000000"
PartitionNameRef="flightControl" Offset="130000000"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="30000000"
PartitionNameRef="flightManagement" Offset="140000000"/>
                <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="10000000"
PartitionNameRef="IOProcessing" Offset="170000000"/>
```

```
               <ar:PartitionTimeWindow PeriodicProcessingStart="false" Duration="20000000"
PartitionNameRef="IVHM" Offset="180000000"/>
       </ar:Schedules>

               <ar:HealthMonitoring>
               <ar:SystemErrors>
                      <ar:SystemError ErrorIdentifier="1" Description="Configuration Error"/>
                      <ar:SystemError ErrorIdentifier="2" Description="Module Config Error"/>
                      <ar:SystemError ErrorIdentifier="3" Description="partition config error"/>
                      <ar:SystemError ErrorIdentifier="4" Description="partition init error"/>
                      <ar:SystemError ErrorIdentifier="5" Description="segmentation error"/>
                      <ar:SystemError ErrorIdentifier="6" Description="time duration exceeded"/>
                      <ar:SystemError ErrorIdentifier="7" Description="invalid OS call"/>
                      <ar:SystemError ErrorIdentifier="8" Description="divide by 0"/>
                      <ar:SystemError ErrorIdentifier="9" Description="floating point error"/>
               </ar:SystemErrors>

               <ar:ModuleHM StateIdentifier="1" Description="module init">
                      <ar:ErrorAction ErrorIdentifierRef="1" ModuleRecoveryAction="SHUTDOWN"/>
                      <ar:ErrorAction ErrorIdentifierRef="2" ModuleRecoveryAction="SHUTDOWN"/>
                      <ar:ErrorAction ErrorIdentifierRef="5" ModuleRecoveryAction="SHUTDOWN"/>
                      <ar:ErrorAction ErrorIdentifierRef="6" ModuleRecoveryAction="IGNORE"/>
                      <ar:ErrorAction ErrorIdentifierRef="7" ModuleRecoveryAction="IGNORE"/>
               </ar:ModuleHM>

               <ar:ModuleHM StateIdentifier="2" Description="system function execution">
                      <ar:ErrorAction ErrorIdentifierRef="5" ModuleRecoveryAction="SHUTDOWN"/>
                      <ar:ErrorAction ErrorIdentifierRef="6" ModuleRecoveryAction="IGNORE"/>
                      <ar:ErrorAction ErrorIdentifierRef="7" ModuleRecoveryAction="IGNORE"/>
                      <ar:ErrorAction ErrorIdentifierRef="8" ModuleRecoveryAction="IGNORE"/>
                      <ar:ErrorAction ErrorIdentifierRef="9" ModuleRecoveryAction="IGNORE"/>
               </ar:ModuleHM>

               <ar:MultiPartitionHM TableName="Avionic partitions HM table">
                      <ar:ErrorAction       ErrorIdentifierRef="3" ErrorLevel="PARTITION" />
                      <ar:ErrorAction       ErrorIdentifierRef="4" ErrorLevel="PARTITION" />
                      <ar:ErrorAction       ErrorIdentifierRef="5" ErrorLevel="PARTITION" />
                      <ar:ErrorAction       ErrorIdentifierRef="6" ErrorLevel="PARTITION" />
                      <ar:ErrorAction       ErrorIdentifierRef="7" ErrorLevel="PARTITION" />
                      <ar:ErrorAction ErrorIdentifierRef="8" ErrorLevel="PARTITION" />
                      <ar:ErrorAction       ErrorIdentifierRef="9" ErrorLevel="PARTITION" />
               </ar:MultiPartitionHM>

               <ar:MultiPartitionHM TableName="System partitions HM table">
                      <ar:ErrorAction ErrorIdentifierRef="3" ErrorLevel="MODULE"
ModuleRecoveryAction="SHUTDOWN" />
                      <ar:ErrorAction ErrorIdentifierRef="4" ErrorLevel="MODULE"
ModuleRecoveryAction="RESET" />
                      <ar:ErrorAction       ErrorIdentifierRef="5" ErrorLevel="PARTITION" />
                      <ar:ErrorAction ErrorIdentifierRef="6" ErrorLevel="MODULE"
ModuleRecoveryAction="RESET" />
                      <ar:ErrorAction       ErrorIdentifierRef="7" ErrorLevel="PARTITION" />
                      <ar:ErrorAction ErrorIdentifierRef="8" ErrorLevel="PARTITION" />
                      <ar:ErrorAction       ErrorIdentifierRef="9" ErrorLevel="PARTITION" />
               </ar:MultiPartitionHM>

               <ar:PartitionHM MultiPartitionHMTableNameRef="System partitions HM table"
TableName="systemManagement HM table">
                      <ar:ErrorAction ErrorIdentifierRef="5" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="MEMORY_VIOLATION" />
                      <ar:ErrorAction ErrorIdentifierRef="7" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="ILLEGAL_REQUEST" />
                      <ar:ErrorAction       ErrorIdentifierRef="8" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="NUMERIC_ERROR" />
                      <ar:ErrorAction       ErrorIdentifierRef="9" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="NUMERIC_ERROR" />
               </ar:PartitionHM>

               <ar:PartitionHM MultiPartitionHMTableNameRef="Avionic partitions HM table"
TableName="flightControls HM table">
                      <ar:ErrorAction ErrorIdentifierRef="3" PartitionRecoveryAction="IDLE"
ErrorLevel="PARTITION" />
                      <ar:ErrorAction       ErrorIdentifierRef="4"
PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PARTITION" />
```

```
                       <ar:ErrorAction ErrorIdentifierRef="5" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="MEMORY_VIOLATION" />
                       <ar:ErrorAction       ErrorIdentifierRef="6"
PartitionRecoveryAction="WARM_RESTART" ErrorLevel="PROCESS" ErrorCode="DEADLINE_MISSED" />
                       <ar:ErrorAction ErrorIdentifierRef="7" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="ILLEGAL_REQUEST" />
                       <ar:ErrorAction       ErrorIdentifierRef="8" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="NUMERIC_ERROR" />
                       <ar:ErrorAction       ErrorIdentifierRef="9" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="NUMERIC_ERROR" />
               </ar:PartitionHM>

               <ar:PartitionHM MultiPartitionHMTableNameRef="Avionic partitions HM table"
TableName="flightManagement HM table">
                       <ar:ErrorAction ErrorIdentifierRef="3" PartitionRecoveryAction="IDLE"
ErrorLevel="PARTITION" />
                       <ar:ErrorAction       ErrorIdentifierRef="4"
PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PARTITION" />
                       <ar:ErrorAction ErrorIdentifierRef="5" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="MEMORY_VIOLATION" />
                       <ar:ErrorAction       ErrorIdentifierRef="6"
PartitionRecoveryAction="WARM_RESTART" ErrorLevel="PROCESS" ErrorCode="DEADLINE_MISSED" />
                       <ar:ErrorAction ErrorIdentifierRef="7" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="ILLEGAL_REQUEST" />
                       <ar:ErrorAction       ErrorIdentifierRef="8" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="NUMERIC_ERROR" />
                       <ar:ErrorAction       ErrorIdentifierRef="9" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="NUMERIC_ERROR" />
               </ar:PartitionHM>

               <ar:PartitionHM MultiPartitionHMTableNameRef="Avionic partitions HM table"
TableName="IO processing HM table">
                       <ar:ErrorAction ErrorIdentifierRef="3" PartitionRecoveryAction="IDLE"
ErrorLevel="PARTITION" />
                       <ar:ErrorAction       ErrorIdentifierRef="4"
PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PARTITION" />
                       <ar:ErrorAction ErrorIdentifierRef="5" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="MEMORY_VIOLATION" />
                       <ar:ErrorAction       ErrorIdentifierRef="6"
PartitionRecoveryAction="WARM_RESTART" ErrorLevel="PROCESS" ErrorCode="DEADLINE_MISSED" />
                       <ar:ErrorAction ErrorIdentifierRef="7" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="ILLEGAL_REQUEST" />
                       <ar:ErrorAction       ErrorIdentifierRef="8"
PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PARTITION"/>
                       <ar:ErrorAction       ErrorIdentifierRef="9"
PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PARTITION"/>

               </ar:PartitionHM>
                       <ar:PartitionHM MultiPartitionHMTableNameRef="Avionic partitions HM table"
TableName="IHVM HM table">
                       <ar:ErrorAction ErrorIdentifierRef="3" PartitionRecoveryAction="IDLE"
ErrorLevel="PARTITION" />
                       <ar:ErrorAction       ErrorIdentifierRef="4"
PartitionRecoveryAction="COLD_RESTART" ErrorLevel="PARTITION" />
                       <ar:ErrorAction ErrorIdentifierRef="5" PartitionRecoveryAction="IDLE"
ErrorLevel="PROCESS" ErrorCode="MEMORY_VIOLATION" />
                       <ar:ErrorAction       ErrorIdentifierRef="6" PartitionRecoveryAction="IDLE"
ErrorLevel="PARTITION"/>
                       <ar:ErrorAction ErrorIdentifierRef="7" PartitionRecoveryAction="IDLE"
ErrorLevel="PARTITION"/>
                       <ar:ErrorAction       ErrorIdentifierRef="8" PartitionRecoveryAction="IDLE"
ErrorLevel="PARTITION"/>
                       <ar:ErrorAction       ErrorIdentifierRef="9" PartitionRecoveryAction="IDLE"
ErrorLevel="PARTITION"/>
               </ar:PartitionHM>

       </ar:HealthMonitoring>

</ar:MODULE>
```

AERONAUTICAL RADIO, INC.
2551 Riva Road
Annapolis, Maryland 24101-7435 USA

SUPPLEMENT 1
TO
ARINC SPECIFICATION 653

AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE
PART 1 – REQUIRED SERVICES

Published: October 16, 2003

Prepared by the Airlines Electronic Engineering Committee

Adopted by the Airlines Electronic Engineering Committee: September 16, 2003

A. PURPOSE OF THIS DOCUMENT

This Supplement introduces various changes and additions to ARINC Specification 653. It provides clarification of the definition of partition management, intrapartition communication and health monitoring. ARINC 653-1 supercedes the original release of the standard (1997). It provides refinement and clarification to the phase 1 standard.

B. ORGANIZATION OF THIS SUPPLEMENT

The first part of this document, printed on goldenrod-colored paper, contains descriptions of the changes introduced in Specification 653 by this Supplement. The second part consists of replacement material, organized by section number, for the Specification to reflect these changes. The modified and added material is identified by the “c-1” symbol in the margins. ARINC Specification 653 was updated by incorporating the replacement material. The goldenrod-colored pages are inserted inside the rear cover of the Specification.

C. CHANGES TO ARINC CHARACTERISTIC 653 INTRODUCED BY THIS SUPPLEMENT

This section presents a complete tabulation of the changes and additions to Specification 653 introduced by this Supplement. Each change or addition is defined by the section number and the title that will be employed when this Supplement is incorporated. In each case, a brief description of the change or addition is included.

1.3.2 Software Decomposition

A system partition was added to enable non-ARINC 653 calls to be made of the O/S. System partition concepts were added to make the O/S modular and remove platform dependencies from the O/S. This makes it easier for the O/S to be developed separately from the processing platform.

1.5 Document Overview

This document is divided into five sections, Introduction, System Overview, Service Requirements, Compliance to the APEX Interface and XML Configuration Tables.

1.6.3 Ada 95

The title of this section was updated to remove references to Ada 9X.

1.6.5 Extensible Markup Language (XML) 1.0

This section was added to describe the Extensible Markup Language (XML) used for ARINC 653 configuration tables.

1.7.4 ARINC 629

This section was deleted by Supplement 1.

1.7.5 ARINC 659

This section was deleted by Supplement 1.

1.7.6 RTCA DO-178 / EUROCAE ED-12

This section was expanded to include EUROCAE reference.

1.7.7 RTCA DO-255 / EUROCAE ED-96

The title of this section was changed and clarifications were made to the current specifications.

1.7.8 RTCA SC-200 and EUROCAE WG-60

A new section was added to describe the ongoing standards work by RTCA and EUROCAE.

2.1 System Architecture

Changes were made to this section to specify that the system integrator is expected to configure the environment to ensure the correct routing of the message. The paragraph referring to Core/Executive (COEX) interface was deleted.

2.2 Hardware

This section was updated to describe the preferred hardware interface. ARINC 659, backplane bus, is referenced as an example. Text was added to state that processor atomic operations might induce jitter on time slicing and should have minimal effect on scheduling.

2.3.1 Partition Management

Clarification of major time frame, start of major time frame, order of partition windows, and periodic release point for a process was provided. The major time frame is defined as a multiple of the least common multiple of all partition periods in the module. Each major frame contains identical partition scheduling windows.

2.3.1.1 Partition Attribute Definition

An additional fixed attribute called System Partition is included. The VARIABLE ATTRIBUTES part is augmented with Start Condition attribute. The description of partition mode is augmented and transferred in the new Section 2.3.1.4.1, Partition Mode Description.

2.3.1.2 Partition Control

This section was updated to discuss control of the core module. The partition is responsible for invoking the appropriate calls to transition from the initialization phase to the operational mode.

2.3.1.3 Partition Scheduling

The Commentary referring to ARINC 659 was deleted.

2.3.1.4 Partition Modes

A new section was added to define partition modes. The SET_PARTITION_MODE service transfers the partitions from one mode to another. Partition modes and transitions are described in a new state diagram.

2.3.1.4.1 Partition Modes Description

New section was added to clarify partition modes.

2.3.1.4.2 Partition Modes Transition

New section was added to discuss how the core module manages transitions of a partition from one mode to another.

2.3.2.2.1 Process State Transitions

New material was added to clarify process states and process transitions, in accordance with partition modes and partition mode transitions. The subordinate sections were also updated.

2.3.3 Time Management

The figure in this section showing replenish on deadline was updated.

2.3.5 Interpartition Communication

This section was clarified and reorganized. Commentary was deleted.

2.3.5.1 Communication Principles

This section was changed to delete the second paragraph and the Commentary. Messages are received atomically, i.e., a partially received message should never be received by the destination port.

2.3.5.2 Message Communication Levels

The Commentary in this section was revised. The reference to ARINC 659 was deleted. All communications (data buses and data networks) are expected to look the same to the APEX interface. References to specific I/O types were deleted.

2.3.5.3.2 Periodic/Aperiodic

The Commentary in this section was expanded to describe the preferred use of periodic and aperiodic messages.

2.3.5.3.3 Broadcast, Multicast and Unicast Messages

Changes were included to add multicast and unicast messages.

2.3.5.4 Message Type Combinations

This section was deleted by Supplement 1 due to potential problems with implementation.

2.3.5.5 Channels and Ports

This section changed to emphasize channels and ports are expected to be entirely defined at system configuration time.

2.3.5.6 Modes of Transfer

This section expanded to say that it is assumed that the underlying system supports the behavior defined herein.

2.3.5.6.1 Sampling Mode

This section was modified to say it is the responsibility of the underlying port implementation to properly handle data segmentation. A partial message is not delivered to the destination port.

2.3.5.6.2 Queuing Mode

This section was revised to support the content of Section 3.6.1, Interpartition Communication Types. The statement that says no message is lost in the queuing mode was deleted. The Commentary referring to segmentation and reassembly was removed. An addition to this section states that a partial message is not delivered to the destination port in the queuing mode.

2.3.5.7.2 Port Name

The example referring to Measured Elevator Position was modified.

2.3.5.7.8 Mapping Requirements

This section was clarified to define the means of external port communications. The additions to this section define the mapping requirement attributes. Mapping ports is the primary means of selecting ports for external I/O. In addition, a port could be mapped to a device driver or “pseudo device driver” by defining a procedure mapping attribute.

2.3.6 Intrapartition Communication

A sentence was added to state that all intrapartition message-passing mechanisms must ensure atomic message access.

2.4 Health Monitor

Section 2.4 and its subordinate sections were completely re-written to define the Health Monitor (HM) and the Error Response Mechanisms. This includes the role of the O/S and the role of a System Partition. The HM is responsible for monitoring and reporting faults and failures in hardware, application software and O/S software. The HM may be made up of O/S components and system partitions. The subjects covered in section are Error Levels, Fault Detection and Response, and Recovery Actions.

2.5.1 Configuration Information Required by the Integrator

References to ARINC 629 were deleted.

2.5.2 Configuration Tables

New material was added to describe the use of Extensible Mark-Up Language (XML) as a standard way of defining configuration tables. This is viewed to be an effective development tool for system integration. The O/S developer and system integrator roles are defined. Appendices G, H and I provide detail and example XML-Schema.

2.5.2.2 Configuration Tables for Inter-Partition Communication

References to ARINC 629 were deleted.

2.5.2.3 Configuration Tables for Health Monitor

This section was updated to describe how the HM tables are constructed from the O/S error codes. A reference was added to recognize Appendix G, an example table generated by the XMLSpy Schema Editor.

2.6 Verification

The title was changed to delete reference to validation. The reference to airframe manufacturer requirements was deleted.

3.1.2 OUT Parameter Value

This section was added to describe value of OUT parameter.

3.2.1 Partition Management Types

The new type START_CONDITION_TYPE used by GET_PARTITION_STATUS was added. A sentence was added to state that the SYSTEM_TIME_TYPE common to many APEX services is defined in Section 3.4.1 of this document.

3.2.2 Partition Managment Services

Title was changed to be consistent with Process Management Services and Time Management Services.

3.2.2.2 SET_PARTITON_MODE

A new condition is added in set partition mode to avoid transition from COLD_START to WARM_START.

3.3.1 Process Management Types

A sentence was added to clarify that SYSTEM_TIME_TYPE is common to many APEX services. SYSTEM_TIME_TYPE is defined in Section 3.4.1.

3.3.2 Process Management Services

Two new services were added in the list of process management services:

DELAYED_START service to allow phasing of the process schedule**.**

GET_MY_ID service, providing the capability to get the identifier of the current process.

3.3.2.3 CREATE PROCESS

The references to dynamic memory allocation were deleted from ARINC 653. The effect of using INFINITE_TIME_VALUE was described.

3.3.2.4 SET PRIORITY

The management of queues for processes waiting on a resource, when the priority changes, was clarified. There was a request for Commentary on this subject, explaining queue management.

3.3.2.5 SUSPEND SELF

A new Commentary was added clarifying the effect of suspend on a self-suspended process.

3.3.2.6 SUSPEND

The effects of SUSPEND on a suspended or self-suspended process was clarified.

3.3.2.7 RESUME

A new test was added prior to the action to RESUME a periodic process, since a periodic process can not be suspended.

3.3.2.8 STOP_SELF

Changes were made to correct an error in this section. If the current process is the error handler process and preemption is disabled, and previous process was stopped, it is not possible to return to the previous process.

3.3.2.9 STOP

Commentary was added to this section.

3.3.2.10 START

This section was expanded to describe two forms of START, and new definitions of START and DELAYED START. The difference between START was clarified for periodic and a periodic processes.

3.3.2.11 DELAYED_START

This section was added to describe a DELAYED START service for processes to enable phasing of a process schedule.

3.3.2.12 LOCK_PREEMPTION

(This section was renumbered.)

3.3.2.13 UNLOCK_PREMPTION

(This section was renumbered.)

3.3.2.14 GET_MY_ID

Adding a new service to provide the identification of the current process. If a procedure is used by several processes, this service allows the procedure to know which process is calling it. The GET_MY_ID service request returns the process identifier of the current process.

3.4.1 Time Management Types

The definition of Time Management was clarified.

3.4.2.2 PERIODIC_WAIT

This section was modified to describe way that process is suspended and the way that it recognizes release point.

3.4.2.4 REPLENISH

This section was modified to allow replenishment with a BUDGET_TIME.

3.6.1 Interpartition Communication Types

A sentence was added to state that the SYSTEM_TIME_TYPE common to many APEX services is defined in Section 3.4.1 of this document.

3.6.2.1 Sampling Port Services

Changes to this section harmonize the parameters of CREATE service with the other objects (buffer and queuing port) and clarify the testing of parameters. This includes modification of the computation of VALIDITY in the GET_SAMPLING_STATUS to have the validity of the last read message. The validity of the current message is provided with the READ_SAMPLING_MESSAGE and consistent with the read message. This section and the subordinate sections were updated to modify the pseudo code to account for address and procedure attributes.

3.6.2.2 Queuing Port Services

Changes to this section clarify the use of QUEUING PORT services and BUFFER services using for both MAX_MESSAGE_SIZE and MAX_NB_MESSAGE. A new statement is added in Section 3.6.2.2.1 whereby CREATE QUEUING PORT creates a queue for the number of maximum size messages. This section and the subordinate sections were updated.

3.7.1 Intrapartition Communication Types

Changes to this section harmonize the declarations, status and parameters of the inter/intra-partition communication services, for example buffer, queuing port, blackboard and sampling port (see Section 3.6.1). A sentence was added to state that the SYSTEM_TIME_TYPE common to many APEX services is defined in Section 3.4.1 of this document.

3.7.2.1 Buffer Services

This section and its subordinate sections were harmonized. The use of QUEUING PORT services and BUFFER services using for both MAX_MESSAGE_SIZE and MAX_NB_MESSAGE.

3.7.2.2 Blackboard Services

This section and subordinate sections were updated to describe blackboard services. A comment was added to clarify the situation where several processes are waiting on an empty blackboard and another process writes in this blackboard.

3.7.2.3 Semaphore Services

Sentence was deleted. The maximum number of semaphores that can be used by a partition is given in the configuration table.

3.7.2.3.2 WAIT_SEMAPHORE

The return code value for NOT_AVAILABLE was changed. Current value of SEMAPHORE_ID<=0 and TIME_OUT=0.

3.7.2.4 Event Services

Sentence was deleted. The maximum number of events that can be used by a partition is given in the configuration table.

3.8 Health Monitoring

This section was updated to clarify the introduction to Health Monitoring (HM).

3.8.1 Health Monitoring Types

This section modifies the maximum length of the message returned by the service GET_ERROR_STATUS. Since the HM and the RAISE_APPLICATION_ERROR are the providers of error messages, the maximum size of error messages of the HM is defined by the O/S, rather than the application. A predefined MAX_ERROR_MESSAGE_SIZE and associated types are defined.

3.8.2 Health Monitoring Services

Health Monitoring was specified in this section and subordinate sections.

4.2 O/S Implementation Compliance

This section was modified to say that a conforming implementation may support additional services or data objects. It may even implement nonstandard extensions. If an O/S does not support the entire APEX standard, these limitations should be clearly documented*.*

5.0 XML Configuration Tables

A new section was added to explain how XML is used to define configuration data independent of the underlying implementation.

APPENDIX A, GLOSSARY

The glossary was updated to include new terms and to provide new definitions for existing terms. These include Ada, Application, Major Frame, Message, Pseudo-partition, Redundancy, Robust Partitioning, Standard Partition and System Partition.

APPENDIX C, APEX SERVICES SPECIFICATION GRAMMAR

Statement was added to state that error code processing is not specified by ARINC 653.

APPENDIX D, ADA INTERFACE SPECIFICATION

Modifications were made to the Ada code to support other changes introduced in Supplement 1. The rationale is included as a preamble to this Appendix.

APPENDIX E, C INTERFACE SPECIFICATION

Modifications were made to the C code to support other changes introduced in Supplement 1. The rationale is included as a preamble to this Appendix.

APPENDIX F, APEX Service Return Code Matrix

This Appendix was deleted by Supplement 1 on the grounds that it contained redundant information.

APPENDIX G, GRAPHICAL VIEW OF ARINC 653 XML- SCHEMA

This Appendix was added to describe the XML-Schema configuration.

APPENDIX H, ARINC 653 XML-SCHEMA

This Appendix was added to include the actual XML-Schema code file.

APPENDIX I, EXAMPLE ARINC 653 XML INSTANCE FILE

An example XML instance file was added for an ARINC 653 implementation.

AERONAUTICAL RADIO, INC.
2551 Riva Road
Annapolis, Maryland 24101-7435 USA

SUPPLEMENT 2
TO
ARINC SPECIFICATION 653

AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE
PART 1 - REQUIRED SERVICES

Published: March 7, 2006

Prepared by the Airlines Electronic Engineering Committee

Adopted by the Airlines Electronic Engineering Committee: October 4, 2005

A. PURPOSE OF THIS DOCUMENT

This Supplement introduces various changes and additions to ARINC Specification 653. It provides clarification of the definition of partition management, initialization and corrections to ARINC 653 pseudo code.

B. ORGANIZATION OF THIS SUPPLEMENT

The first part of this document contains descriptions of the changes introduced in ARINC 653 by this Supplement. The second part consists of replacement material, organized by section number, for the Specification to reflect these changes. ARINC 653 will be updated by incorporating the replacement material. The goldenrod-colored pages are inserted inside the rear cover of the Specification.

C. CHANGES TO ARINC SPECIFICATION 653 INTRODUCED BY THIS SUPPLEMENT

This section presents a complete tabulation of the changes and additions to ARINC 653 introduced by this Supplement. Each change or addition is defined by the section number and the title that will be used when this Supplement is incorporated. In each case, a brief description of the change or addition is included.

1.2 Scope

The section on document scope was added by Supplement 2. Section 1 was re-numbered accordingly.

1.3 APEX Phases

Commentary was expanded to introduce three parts of ARINC 653.

1.6 Document Overview

Information was added to describe contents of Parts 1, 2, and 3 of ARINC 653.

2.3.1 Partition Management

The second paragraph was modified for clarity regarding support for robust partitioning.

2.3.1.4.1 Partition Modes Description

The definition of IDLE was clarified to indicate that resources (memory for example) are consumed. The definition of COLD_START and WARM_START was clarified to indicate that pre-emption is disabled during these modes of operation. This clarification was done in order to align with the use of “pre-emption disabled” within Section 3 pseudo code.

2.3.1.4.1.1 COLD_START and WARM_START Considerations

Commentary was expanded to describe activities in the initialization phase.

2.3.1.4.2.2 Transition Mode Description

The current ARINC Specification 653 Section 2.3.1.4.2.2 (2), says that COLD_START -> WARM_START is not allowed. However, Section 2.3.1.4.2.2 (3a) says that Cold_Start -> Idle is allowed, and Section 2.3.1.4.2.2 (6b) says that Idle -> Warm_Start is allowed. This allows a transition from COLD_START -> WARM_START via a transition through the idle state.

Words were added to state that a transition from COLD_START -> WARM_START through IDLE is not allowed.

2.3.5.6.1 Sampling Mode

The wording that states that only fixed length messages are allowed in the sampling mode was removed. This commentary is not consistent with the sampling port API services. The sampling port API services support the transmission of variable length messages up to the MAX_MESSAGE_SIZE.

Also removed commentary regarding broadcast of messages.

2.3.5.8 Port Control

The wording that described OS generated time stamps for queuing ports was removed, because it was inconsistent with Section 3.6.2.2 of the document. The queuing port APIs do not return a time stamp.

The wording that states that only fixed length messages are allowed in the sampling mode was removed. This commentary is not consistent with the sampling port API services. The sampling port API services support the transmission of variable length messages up to the MAX_MESSAGE_SIZE.

The specification uses the term “age of the copied message”, but no definition for this is given. A definition of “age” was added to define the age of a copied message as “the difference between the value of the system clock (when READ_SAMPLING_MESSAGE is called) and the system clock when the OS copied the messages from the channel into the destination port”.

2.4.2.1 Process Level Error Response Mechanisms

An example was added for the use of a process level power fail error condition.

A description of the dependency between the process level error handling and the system integrator was added.

2.5.2.1 Configuration Tables for System Initialization

The words “compatible with the configuration” were added to indicate that validity is checked against the configuration data.

3.3.2.6 SUSPEND

The current specification would allow a process to be suspended even though it had called LOCK_PREEMPTION.

- This condition could happen in the case where the process that had disabled preemption generated an error that invoked the process level error-handler, and then the process level error_handler attempted to SUSPEND the process that had disabled preemption.
- This could also happen in the case where a process had disabled preemption, and then the deadline for another process expired. In this case, the process level error-handler should be invoked to handle the deadline expiration, and it might attempt to SUSPEND the process that had disabled preemption.

- The behavior after the process is suspended is undefined. One might assume that the OS would either schedule no processes or it would let the next highest priority process run. In this case, preemption would still be disabled, even though the running process would not know that preemption was disabled.

The SUSPEND service was modified to behave analogously SUSPEND_SELF. That is, SUSPEND will return INVALID_MODE if preemption is disabled and the process id of the process that is being suspended is the process holding the preemption lock.

3.3.2.8 STOP_SELF

This section was expended to describe the behavior of STOP_SELF when the partition is in the WARM_START or COLD_START modes.

3.3.2.12 LOCK_ PREEMPTION

The UNLOCK_PREEMPTION pseudo code was corrected to account for when it is called during COLD_START or WARM_START modes and when it is called from within the error_handler process.

3.3.2.13 UNLOCK_ PREEMPTION

The UNLOCK_PREEMPTION pseudo code was corrected to account for when it is called during COLD_START or WARM_START modes and when it is called from within the error_handler process.

3.4.2.4 REPLENISH

The REPLENISH pseudo code was corrected to account for when it is called during COLD_START or WARM_START modes and when it is called from within the error_handler process.

3.6.2.1 Sampling Port Services

The wording that stated that only fixed length messages are allowed in the sampling mode was removed. This commentary is not consistent with the sampling port API services. The sampling port API services support the transmission of variable length messages up to the MAX_MESSAGE_SIZE.

3.6.2.1.1 CREATE_SAMPLING_PORT

In the pseudo code, modified three sentences using the words “compatible with configuration” to be explicit.

In the return code descriptions modified the three occurrences of the word “compatible with configuration” to match the new pseudo code.

3.6.2.1.2 WRITE_SAMPLING_MESSAGE

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

In the pseudo code, modified the sentence using the words “compatible with configuration” to be explicit.

In the return code description, modified the occurrence of the word “compatible with configuration” to match the new pseudo code.

3.6.2.1.3 READ_SAMPLING_MESSAGE

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

3.6.2.2.1 CREATE_QUEUING_PORT

In the pseudo code, modified three sentences using the words “compatible with configuration” to be explicit.

In the return code, descriptions modified the three occurrences of the word “compatible with configuration” to match the new pseudo code.

3.6.2.2.2 SEND_QUEUING_MESSAGE

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

In the pseudo code, modified the sentence using the words “compatible with configuration” to be explicit.

In the return code, description modified the occurrence of the words “compatible with configuration” to match the new pseudo code.

3.6.2.2.3 RECEIVE_QUEUING_MESSAGE

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

3.7.2.1.2 SEND_BUFFER

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

3.7.2.1.3 RECEIVE_BUFFER

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

3.7.2.2.2 DISPLAY_BLACKBOARD

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

3.7.2.2.3 READ_BLACKBOARD

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

3.7.2.3.2 WAIT_SEMAPHORE

Correction was made by replacing the word null with zero.

3.8.2.1 REPORT_APPLICATION_MESSAGE

This section expanded by adding a clarification on the use of MESSAGE_ADDR.

3.8.2.2 CREATE_ERROR_HANDLER

This section expanded by adding a clarification for the case when the error handler calls LOCK_PREEMPTION and UNLOCK_PREEMPTION.

3.8.2.4 RAISE_APPLICATION_ERROR

The RAISE_APPLICATION_ERROR service was essentially reverted back to the manner in which it was specified in the 1997 specification. Additional commentary was added to reflect the behavior when an error handler does not exist. Additional commentary was added to clarify that the HM system can determine the source of an ERROR_CODE (e.g., a process with in a Partition versus the OS) by the use of the SYSTEM STATE attribute that is generated by the OS.

4.2 COMPLIANCE TO APEX INTERFACE

This section was completely revised to introduce compliance and Part 3 of ARINC 653.

4.3 Application Compliance

The last sentence of this section was deleted.

Appendix D

Appendix D was re-organized to include Ada95 bindings in addition to the Ada83 bindings provided in Supplement 1. Appendix D now includes D.1 (Ada83) and D.2 (Ada95).

The current specification of string representation in Appendix D is inconsistent. The new specification for NAME_TYPE (string representation) clarifies the following:

- The definition of ARINC 653 NAME_TYPE is based on an 'n-character array (i.e. fixed length).
- To avoid unnecessary complexity in the underlying operating system, there are no restrictions on the allowable characters. However, it is recommended that users limit themselves to printable characters. The OS should treat the contents of NAME_TYPE objects as case insensitive.
- The use of a NULL character is not required in an object of NAME_TYPE. If a NAME_TYPE object contains NULL characters, the first NULL character in the array of characters should be considered to define the end of the name.

Commentary was added to indicate that the value of constants and certain names are implementation dependent. This makes appendix D consistent with Appendix E.

A “mod 8” clause was added to some of the record structures for alignment purposes.

Removed range constraint on all ID types.

Removed note “warm_start = cold_start”.

Appendix E

The current specification of string representation in appendix E is inconsistent. The new specification for NAME_TYPE (string representation) clarifies the following:

- The definition of ARINC 653 NAME_TYPE is based on an 'n-character array (i.e. fixed length).
- To avoid unnecessary complexity in the underlying operating system, there are no restrictions on the allowable characters. However, it is recommended that users limit themselves to printable characters. The OS should treat the contents of NAME_TYPE objects as case insensitive.
- The use of a NULL character is not required in an object of NAME_TYPE. If a NAME_TYPE object contains NULL characters, the first NULL character in the array of characters should be considered to define the end of the name.

Removed note “warm_start = cold_start”.

Appendix G

Added a note under Figure 5 clarifying why the units for RefreshPeriodSeconds were selected.

The schema file “A653_Part_1_revA.xsd” has updates that are not completely backward compatible with previous XML files, based on the original schema. The incompatibility is that any hex values now need to be preceded by “0x”. All other changes are backward compatible with previous instance files.

The changes to the schema are:

1. Window_Sched_Ext was required (in error). Changed 1..oo to 0..oo to make Window_Sched_Ext optional.
2. Added optional ModuleId to ARINC_653_Module element of new type IdentifierValueType. (see below)
3. Added two simpleType definitions:
    - DecOrHexValueType (allow use of hex 0x[numbers] and decimal numbers.)
    - IdentifierValueType (restricts identifiers to be of DecOrHexValueType)
4. Changed the following attributes to IdentifierValueType
    - All System_State_Entry elements (3) attribute SystemState
    - Error_ID_Level element attribute ErrorIdentifier
    - PartitionType definition attribute ParititionIdentifier
    - Partition_Memory element attribute PartitionIdentifier
    - Partition_Schedule element attribute PartitionIdentifier
    - Window_Schedule element attribute WindowIdentifier
    - Partition_HM_Type definition attribute PartitionIdentifier
    - Channel element attribute ChannelIdentifier
    - Standard_Partition element attribute PartitionIdentifier
5. Change the following attributes to DecOrHexValueType:
    - Memory_Requirements element attributes SizeBytes and PhysicalAddress
    - PortType definition attribute MaxMessageSize
    - Pseudo_Partition element attribute PhysicalAddress
    - Standard_Partition element attribute PhysicalAddress
    - ProcessElement element attribute StackSize

Appendix H

Replaced the entire schema with Rev A version of schema.

Appendix I

The example “instance file” was changed so that it would validate against the new schema. All places where an xs:hexBinary value was entered a value that starts with “0x”.

The schema file location was changed to work with the updated A653_Part_1_revA.xsd file.

AERONAUTICAL RADIO, INC.
2551 Riva Road
Annapolis, Maryland 24101-7435

SUPPLEMENT 3
TO
ARINC SPECIFICATION 653

AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE
PART 1 – REQUIRED SERVICES

Published: November 15, 2010

Prepared by the AEEC

Adopted by the AEEC Executive Committee: October 6, 2010

A. PURPOSE OF THIS DOCUMENT

This Supplement introduces various changes and additions to ARINC Specification 653. It provides clarification of the definition of Health Management and configuration Data defined in extensible Markup Language (XML).

B. ORGANIZATION OF THIS SUPPLEMENT

This document contains descriptions of the changes introduced in ARINC 653 by this Supplement. What follows is a complete version of ARINC 653 Part 1. Technical changes introduced by Supplement 3 are highlighted in blue bold text.

C. CHANGES TO ARINC SPECIFICATION 653 INTRODUCED BY THIS SUPPLEMENT

This section presents a complete tabulation of the changes and additions to ARINC 653 introduced by this Supplement. Each change or addition is defined by the section number and the title that will be used when this Supplement is incorporated. In each case, a brief description of the change or addition is included.

1.4.2 Software Decomposition

Figure 1.2, Core Module Component Relationship, was deleted by Supplement 3.

1.5.2 Language Considerations

Last paragraph revised to refer to a particular core module rather than a particular core processor module.

1.6 Document Overview

The reference to Appendix C, APEX Services Specification Grammar, was shown to be deleted by Supplement 3. The reference to Appendix H was updated to refer to XML Schema Types. The reference to Appendix I, Example XML Schema and Corresponding XML Data Instance was added.

2.1 System Architecture

Clarifications made to emphasize that ARINC 653 applies to both traditional federated avionics and Integrated Modular Avionics (IMA). Commentary was added to clarify the term “core software”. Commentary was added to describe the role of ARINC 653 in a multi-processor environment.

2.3 System Functionality

The second paragraph and Commentary was added to describe operation during module power-up and initialization.

2.3.1 Partition Management

The section and its subordinate sections were updated to clarify the role of application software, modules and partitions. Partition attribute definitions were updated. Partition operating modes were clarified.

2.3.2 Process Management

Process management and process control was clarified. Processes within a partition will share the same address space. The definition of process state transitions was updated for clarity. Process scheduling was updated to define behavior in the presences of multiple applications and priority.

2.3.3 Time Management

Commentary was added to describe process deadlines and behavior in the event of a missed deadline.

2.3.4 Memory Management

This section was updated to describe memory management during system configuration and initialization.

2.3.5 Interpartition Communication

The interpartition communication definitions were updated to facilitate communications between ARINC 653 application partitions residing on the same module or on different modules, as well as communications between an ARINC 653 application partition.

2.3.6 Intrapartition Communication

This section was expanded. Attributes associated with intrapartition communications are specified by the application when invoking the corresponding create operation.

As with process and port creation, the amount of memory required to create intrapartition communication mechanisms is allocated from the partition’s memory resources, which is defined at system build time. Processes can create as many intrapartition communication mechanisms as the partition’s memory resources will support.

2.4 Health Monitor

Section 2.4 and its subordinate sections were completely revised. Health Monitor (HM) functions are responsible for responding to and reporting hardware, application, and O/S software faults and failures. ARINC 653 supports HM by providing HM configuration tables and an application level error handler process.

The HM helps to isolate faults and to prevent failures from propagating. Components of the HM are contained within the following software elements:

- O/S – All HM implementations will use the ARINC 653 O/S. The O/S uses the HM configuration tables to respond to pre-defined faults.
- Application Partitions – Where faults are system specific and determined from logic or calculation, application partitions may be used, passing fault data to the O/S via the HM service calls or to an appropriate system partition.
- System Partitions – System partitions can be used by the platform integrators for error management.

2.5 Configuration Considerations

The definition of configuration data to support Health Monitoring (HM) was added. The configuration data is specified in the form of XML-Schema Types. This is elaborated in Section 5 as well as Appendices G, H and I.

2.6 Verification

Figure 2.5.2.3-1, HM Configuration Table Usage, was deleted by Supplement 3.

3.1 Service Request Categories

This section was expanded to support multi-process environments. In order to preserve the integrity of information managed by the services, the requesting process is assumed to never be preempted during the execution of a service, except at the scheduling points which are explicitly mentioned in the semantic description. Ada and C language specifications for the types, records, constants, and service request interfaces used in this section are defined in Appendices D and E.

3.2 Partition Management

This section was updated to describe specifications related to partition management. The pseudo code was updated. For example, if the Operation Mode is Normal, then set to Ready all previously started aperiodic processes, set release point of all previously delayed started aperiodic processes to the system clock time plus their delay times, and set first release points of all previously started periodic processes to the partition's next periodic processing start.

3.3 Process Management

This section and its subordinate sections were updated to define specifications related to process management. The scope of a process management service is restricted to a partition.

3.4 Time Management

This section and its subordinate sections were updated to better define temporal provisions. The Time Management services provide a means to control periodic and aperiodic processes. For periodic processes, each process within a partition is able to specify a maximum elapsed time for executing its processing cycle. This time capacity is used to set a processing deadline time.

3.5 Memory Management

Several clarifications were made to this section. There are no memory management services in ARINC 653, because partitions and their associated memory spaces are defined as part of system configuration activities.

3.6 Interpartition Communication

This section and its subordinate sections were updated to better define interpartition communication. This includes Sampling Port and Queuing Port services.

3.7 Intrapartition Communication

This section and its subordinate sections were updated to better define intrapartition communication. Two communication mechanisms are available for intrapartition communication. The first mechanism allows inter-process communication and synchronization via partition-defined buffers and blackboards. The second allows inter-process synchronization via partition-defined counting semaphores and events.

3.8 Health Monitoring

The Health Monitoring (HM) services were updated. The HM is invoked by an application calling the RAISE_APPLICATION_ERROR service or by the O/S or hardware detecting a fault. The recovery action is dependent on the error level. The recovery actions for each module and each partition level error are specified in the

HM tables. The recovery actions for process level errors are defined by the application programmer in a special error handler process.

4.0 COMPLIANCE TO APEX INTERFACE

There were no changes to Section 4.

5.0 XML Configuration Specification

This section was completely revised by Supplement 3. It defines the mechanism for specifying configuration data, in a standard format, that does not depend on the implementation of the ARINC 653 core software.

The ARINC 653 XML-Schema types define the structures and the types of the data needed to configure any ARINC 653 O/S. These XML-Schema types are used by the ARINC 653 O/S implementer to create the XML-Schema that is specific to the O/S implementation. The XML-Schema is used by the system integrator to write the XML configuration.

The ARINC 653 XML-Schema types cannot be altered by the O/S implementer. They may be extended by including them in larger types, if necessary, to incorporate extra configuration types that are specific to the O/S implementation. An example of how ARINC 653 XML-schema types can be extended is provided in Appendix I.

Appendix A, Glossary

The Glossary was updated to include new terms introduced by Supplement 3.

Appendix B Acronyms

No changes made by Supplement 3.

Appendix C

This Appendix deleted by Supplement 3.

Appendix D.1 ADA 83 INTERFACE SPECIFICATION

This Appendix was prepared in two parts, D.1 and D.2, with a description of the roles of each appendix. This is an Ada specification for the APEX interface, compatible with Ada 83 or Ada 95, and derived from Section 3 of ARINC 653 Part 1.

The declarations of the services given below are taken from the standard as are the enumerated types and the names of the others types. Unless specified as implementation dependent, the values specified in this appendix should be implemented as defined.

APPENDIX D.2 ADA 95 INTERFACE SPECIFICATION

This Appendix was prepared in two parts, D.1 and D.2, with a description of the roles of each appendix. This is an Ada specification for the APEX interface, compatible with Ada 83 or Ada 95, and derived from Section 3 of ARINC 653 Part 1.

All types have a defined representation clause to enable compatibility between the C and Ada 95 specifications. The objective is to have the same definitions and representations of the interface in both the Ada and C languages. The specifications are aligned in the same order and define the same items.

Appendix E ANSI C INTERFACE SPECIFICATION

This appendix was updated. This is a compilable ANSI C specification of the APEX interface, derived from Section 3 of ARINC 653 Part 1. The declarations of the services are taken from the standard, as are the enum types and the names of the others types. Unless specified as implementation dependent, the values specified in this appendix should be implemented as defined.

The ANSI C specification follows the same structure (package) as the Ada specification. The objective is to have the same definitions and representations of the interface in both the Ada and C languages. The two specifications are aligned in the same order and define the same items.

Appendix F APEX SERVICE RETURN CODE MATRIX

This appendix was deleted by Supplement 3 to support the new Health Monitoring definition.

Appendix G GRAPHICAL VIEW OF ARINC 653 XML-SCHEMA TYPES

This appendix was completely revised by Supplement 3.

Appendix H ARINC 653 XML-SCHEMA TYPES

This appendix was completely revised by Supplement 3.

Appendix I ARINC 653 XML-SCHEMA TYPE USAGE EXAMPLES

This appendix was added by Supplement 3.

ARINC Standard – Errata Report

1. Document Title

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

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 fax +1 410-266-2047 or standards@arinc.com

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 IA Project Initiation/Modification (APIM)

1.0 Name of Proposed Project **APIM #: _____**

(Insert name of proposed project.)

1.1 Name of Originator and/or Organization

(Insert name of individual and their organization that initiated the APIM)

2.0 Subcommittee Assignment and Project Support

2.1 Suggested AEEC Group and Chairman

(Identify an existing or new AEEC group.)

2.2 Support for the activity (as verified)

Airlines: (Identify each company by name.)

Airframe Manufacturers:

Suppliers:

Others:

2.3 Commitment for Drafting and Meeting Participation (as verified)

Airlines:

Airframe Manufacturers:

Suppliers:

Others:

2.4 Recommended Coordination with other groups

(List other AEEC subcommittees or other groups.)

3.0 Project Scope (why and when standard is needed)

3.1 Description

(Insert description of the scope of the project.)

3.2 Planned usage of the envisioned specification

Use the following symbol to check yes or no below. ☒

New aircraft developments planned to use this specification yes □ no □

Airbus: (aircraft & date)

Boeing: (aircraft & date)

Other: (manufacturer, aircraft & date)

Modification/retrofit requirement yes □ no □

Specify: (aircraft & date)

Needed for airframe manufacturer or airline project yes □ no □

Specify: (aircraft & date)

Mandate/regulatory requirement yes ❑ no ❑

Program and date: (program & date)

Is the activity defining/changing an infrastructure standard? yes ❑ no ❑

Specify (e.g., ARINC 429)

When is the ARINC standard required?
______(month/year)__________

What is driving this date? _______(state reason)_____________________

Are 18 months (min) available for standardization work? yes ❑ no ❑

If NO please specify solution: _________________

Are Patent(s) involved? yes ❑

If YES please describe, identify patent holder: _________________

3.3 Issues to be worked

(Describe the major issues to be addressed.)

4.0 Benefits

4.1 Basic benefits

Operational enhancements yes ❑ no ❑

For equipment standards:

a. Is this a hardware characteristic? yes ❑ no ❑

b. Is this a software characteristic? yes ❑ no ❑

c. Interchangeable interface definition? yes ❑ no ❑

d. Interchangeable function definition? yes ❑ no ❑

If not fully interchangeable, please explain: _________________

Is this a software interface and protocol standard? yes ❑ no ❑

Specify: _________________

Product offered by more than one supplier yes ❑ no ❑

Identify: (company name)

4.2 Specific project benefits (Describe overall project benefits.)

4.2.1 Benefits for Airlines

(Describe any benefits unique to the airline point of view.)

4.2.2 Benefits for Airframe Manufacturers

(Describe any benefits unique to the airframe manufacturer’s point of view.)

4.2.3 Benefits for Avionics Equipment Suppliers

(Describe any benefit unique to the equipment supplier’s point of view.)

5.0 Documents to be Produced and Date of Expected Result

One/Two Project Papers are expected to be completed per the table below.

5.1 Meetings and Expected Document Completion

The following table identifies the number of meetings and proposed meeting days needed to produce the documents described above.

Activity	Mtgs	Mtg-Days (Total)	Expected Start Date	Expected Completion Date
Document a	*# of mtgs*	*# of mtg days*	*mm/yyyy*	*mm/yyyy*
	*# of mtgs **	*# of mtg days **		
Document b	*# of mtgs*	*# of mtg days*	*mm/yyyy*	*mm/yyyy*
	*# of mtgs **	*# of mtg days **		

* Indicate unsupported meetings and meeting days, i.e., technical working group or other ad hoc meetings that do not requiring IA staff support.

6.0 Comments

(Insert any other information deemed useful to the committee for managing this work.)

6.1 Expiration Date for the APIM

April/October 20XX

For IA staff use only

Date Received: Jan 15, 2010 IA staff : ____________

Potential impact: ______

*(**A**. Safety **B**. Regulatory **C**. New aircraft/system **D**. Other)*

Resolution: _______

Authorized, Deferred, Withdrawn, More Detail Needed, Rejected)

Assigned to SC/WG:________________